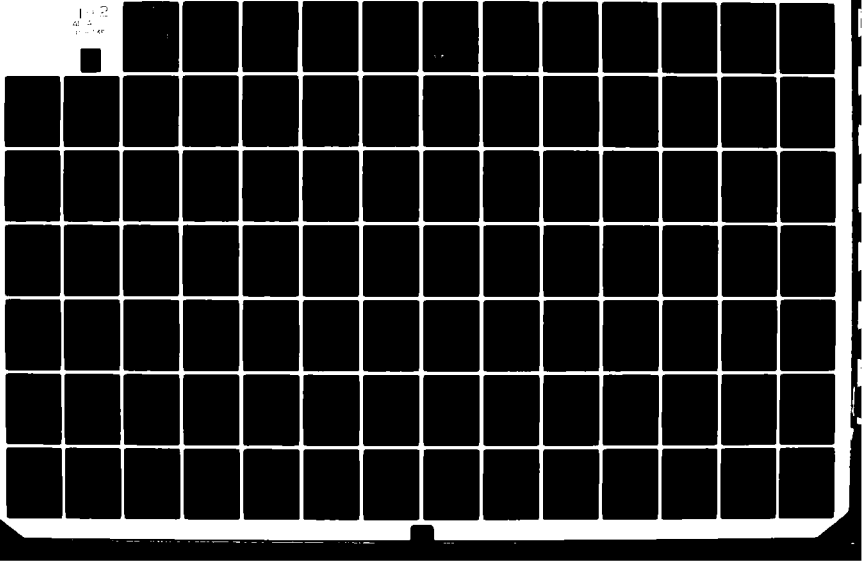
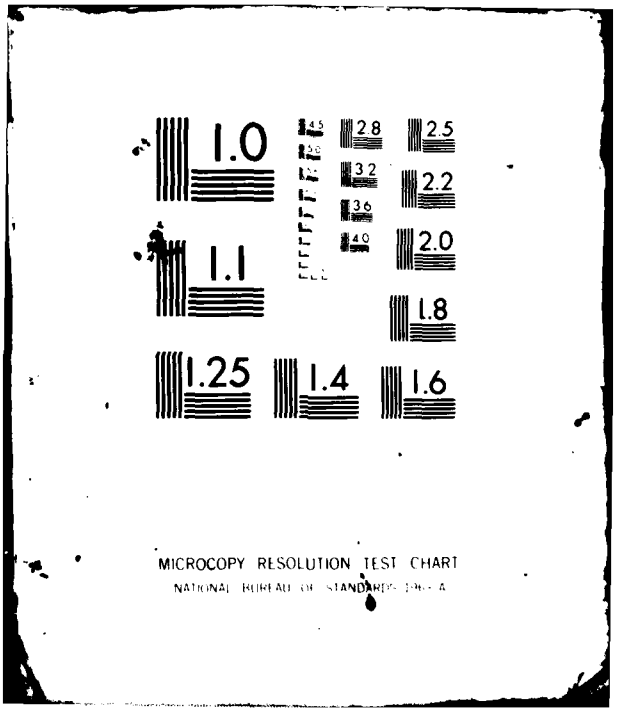
AD-A108 735 STANFORD UNIV CA DEPT OF COMPUTER SCIENCE F/G 6/4
QUERY OPTIMIZATION BY SEMANTIC REASONING.(U)
MAY 81 J J KING N00039-80-G-0132
UNCLASSIFIED STAN-CS-81-857 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

May 1981

(2) LEVEL II

Report. No. STAN-CS-81-857

Query Optimization by Semantic Reasoning

by

Jonathan Jay King

Research sponsored in part by

National Science Foundation
International Business Machines
Defense Advanced Research Projects Agency

Department of Computer Science

Stanford University
Stanford, CA 94305

Query Optimization by Semantic Reasoning

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By

Jonathan Jay King

May 1981

Abstract

The problem of *database query optimization* is to select an efficient way to process a query expressed in logical terms from among the alternative ways it can be carried out in the physical database. This thesis presents a new approach to this problem, called *semantic query optimization.* The goal of semantic query optimization is to produce a semantically equivalent query that is less expensive to process than the original query.

Semantic query optimization actually transforms the original query into a new one by means of a process of inference. The transformations are limited to those that yield a semantically equivalent query, one that is guaranteed to produce the same answer as the original query in any permitted state of the database. This guarantee is achieved because the knowledge used to transform a query is the same knowledge used to insure the *semantic integrity* of the data stored in the database. Thus, semantic query optimization brings together the apparently separate research areas of query processing and database integrity.

The thesis also addresses an important issue in current automatic planning research: production not just of a correct solution but of a "good" one, by means of an efficient problem solver. Semantic query optimization advances the notion of a *problem reformulation* step for problem-solving programs. In this step, equivalent statements of the original problem are sought, one of which may have a better solution than the original problem. This method avoids explicit and possibly costly analysis of efficiency factors during planning itself.

Semantic query optimization can also be viewed as one aspect of *intelligent database mediation.* It applies knowledge of a problem domain and of the capabilities and limitations of the database to pose the most effective and easily processed queries to solve a user's problem.

The thesis formally defines transformations that preserve semantic equivalence for queries in the relational calculus. In addition, it identifies several classes of cost-reducing query transformations for relational database queries, and provides quantitative estimates of the improvements they can produce, based upon widely accepted models of query processing.

The thesis also discusses the design and implementation of a system that carries out semantic query optimization for an important class of relational database queries. The system is called QUIST, standing for **QUery Improvement through Semantic Transformation.**

The QUIST system has analyzed a range of queries for which different transformations apply. For these queries, QUIST obtains substantial reductions in the cost of processing at a negligible cost for the analysis itself.

Acknowledgments

It is a pleasure to acknowledge the help and support I have received from many people in the efforts that have culminated in this thesis.

Gio Wiederhold is responsible for creating the Knowledge Base Management Systems Project within which I have carried out this research. He has consistently fostered the flow of ideas between artificial intelligence and database research that underlies the thesis. His special concern for the effective performance of intelligent computer programs helped to set the direction of my research. Gio never stints in his personal and professional support of his colleagues. I am honored to be among them.

Throughout my years at Stanford, Bruce Buchanan has been a scientific inspiration and a fund of common sense. I marvel at the breadth of his interests and at his ability to locate the heart of a problem. He is consistent, thoughtful, and above all, a gentleman. He has helped me more than once to stay the course.

Of Earl Sacerdoti's many sterling attributes, I want to thank him most for his enthusiasm. Earl strives for connection, extension, and application. He has often persuaded me of the significance of ideas that I have not fully appreciated. Earl is also responsible for much of what clarity of expression appears in the text.

I have had other valuable research colleagues in pursuing this work. Daniel Sagalowicz has spent a great deal of time with me, planting the seeds of many ideas and letting me discover the answers he already knows. Barbara Grosz has helped me in many ways, with a critical eye and a warm heart. Ed Feigenbaum has contributed his unique perspective on the care and feeding of expertise in computer programs. Jerry Kaplan, the ramrod of the KBMS Project, has kept me pointed in the right direction.

Other members of the KBMS Project have been extremely helpful. The roster past and present includes Bing-quan Chen, Jim Davidson, Ramez El-Masri, Sheldon Finkelstein, Hector Garcia, Mohammed Olumi, Tom Rogers, Neil Rowe, David Shaw, and Kyu-Young Whang. Special credit goes to Shel Finkelstein for surviving as my office mate and for trying to add some rigor to my thinking. And of course, Jayne Pickering has kept the whole menagerie in line. Thanks, Jayne, we needed that.

Other friends far and near have helped improve my ideas and have offered support of all kinds. They include Saul Amarel, Bill Baker, Avron Barr, Dave Barstow, Sylvia Bates, Jim Bennett, Denny Brown, Jake Brown, Harold Brown, Pat Burbank, Mark Cartun, Vint Cerf, Mike Clancy, Steve Crocker, Nancy Davis, Randy Davis, Bob Engelmore, Larry Fagin, Dick Gillam, Abra Greenspan, Pat Guiteras, Norm Haas, Doug Hofstadter, Elaine Kant, Peggy Karp, Fred Lakin, Ruth Andrea Levinson, Paul Martin, Larry Masinter, Thorne McCarty, Brian McCune, Charles Mingus, Bernard Mont-Reynaud, Bob Moore, Jack Mostow, Malka Rosen, Betty Scott, Ted Shortliffe, Bob Sproull, Lee Sproull, Mark Stefik, Blair Stewart, Jacquie Stewart, Jacobo Valdes, Bill van Melle, Richard Waldinger, Dave Wilkins, Terry Winograd, Bill Yamamoto, and Ignacio Zabala.

And of course, Luis Trabb-Pardo.

I have five special debts to acknowledge. I want to thank Belvin Williams for hiring me on faith, B.O.Koopman for showing me the work of the scientist in the world, Bill Raub for putting me on the path toward scientific research, Ed Feigenbaum for giving me a chance to straighten things out, and Bob Taylor for refusing to help me quit.

I wish to thank the National Science Foundation, the International Business Machines Corporation, and the Defense Advanced Research Projects Agency for their financial support. This document was composed on the facilities of the Stanford Artificial Intelligence Laboratory and was printed on equipment donated to Stanford by the Xerox Corporation. The research described here was conducted as part of the Knowledge Base Management Systems Project at Stanford University and SRI International, supported by the Advanced Research Projects Agency of the Department of Defense under contract N00039-80-G-0132.

My parents and my sister Stephanie have helped me in every way imaginable. I hope they realize how much.

And thanks most of all to Jonah and to Ellen, for everything.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 Overview of the thesis

The problem of *database query optimization* is to select an efficient way to process a query expressed in logical terms from among the alternative ways it can be carried out in the physical database. This thesis presents a new approach to this problem, called *semantic query optimization* (SQO).

> *The goal of semantic query optimization is to produce a semantically equivalent query that is less expensive to process than the original query.*

Semantic query optimization is a response to inherent limitations in what may be termed *conventional query optimization* methods† ([Selinger79] [Yao79] [Youssefi78]). These methods seek to exploit efficient paths in the physical database. However, it is not possible to supply physical support for all logical relationships because of the high cost to maintain that support when the database is updated. Thus, there will be many queries that both involve access to much data, and in which the logical relationships are not well supported physically. These queries are expensive to process, and conventional techniques are ineffective.

Semantic query optimization actually transforms the original query into a new one by means of a process of inference. The transformations are limited to those that yield a semantically equivalent query, one that is guaranteed to produce the same answer as the original query in any permitted state of the database. This guarantee is achieved because the knowledge used to transform a query is the same knowledge used to insure the *semantic integrity* [McLeod76] or meaningfulness of the data stored in the database. Thus, semantic query optimization brings together the apparently separate research areas of query processing and database integrity.

SQO also addresses an important issue in current automatic planning research: production not just of a correct solution but of a "good" one, by means of an efficient problem solver.

---

†The term *optimization* is a misnomer; there is no claim that the least expensive processing method is found. However, the term is firmly established in the literature.

Semantic query optimization advances the notion of a problem reformulation step for problem-solving programs. In this step, equivalent statements of the original problem are sought, one of which may have a better solution than the original problem. This method avoids explicit and possibly costly analysis of efficiency factors during planning itself.

Semantic query optimization can also be viewed as one aspect of *intelligent database mediation.* It applies knowledge of the problem domain and of the capabilities and limitations of the database to pose the most effective and easily processed queries to solve a user's problem.

As with most query optimization work, the research presented here deals with queries using the relational model of data ([Codd70], [Kim79]).† The thesis formally defines transformations that preserve semantic equivalence for queries in the relational calculus [Codd71]. In addition, it identifies several classes of cost-reducing query transformations for relational database queries, and provides quantitative estimates of the improvements they can produce, based upon widely accepted models of query processing.

The thesis also discusses the design and implementation of a system that carries out semantic query optimization for an important class of relational database queries. The system is called **QUIST**, standing for **QUery Improvement through Semantic Transformation.**

The QUIST system has analyzed a range of queries for which different transformations apply in the context of a simplified query processing model based on the System R access path selector [Selinger79]. For these queries, QUIST's overhead is negligible compared to the estimated reduction of query processing cost. The overhead is also negligible in cases where QUIST determines that there are no constraint targets, or that the query conditions are not satisfiable. The latter condition is detected without recourse to actual data, in contrast to a similar function performed by so-called "cooperative response" systems ([Kaplan79], [Janas79]).

QUIST uses *heuristics* to guide the process of inference that produces equivalent queries. The process is directed toward the application of one or more specific types of transformations on the relational query, such as the elimination of a relation or the introduction of a constraint on an indexed attribute. The only inferences that take place are those that may produce a query that is more efficient to process.

QUIST's inference-guiding heuristics reflect the *expert knowledge* of relational database structure and query processing developed in recent query optimization research. Indeed, it is the existence of fairly wide agreement about models of query processing and optimization issues in the relational setting that makes that setting a a suitable one for exploring semantic query optimization.

The operation of QUIST can be contrasted with that of a conventional query optimizer. A conventional optimizer (Figure 1-1) takes the given query as its input. Its output is a plan consisting of a sequence of retrieval operations in the physical database.

---

†The work is also applicable to other data models particularly where implementations include some fast access paths.

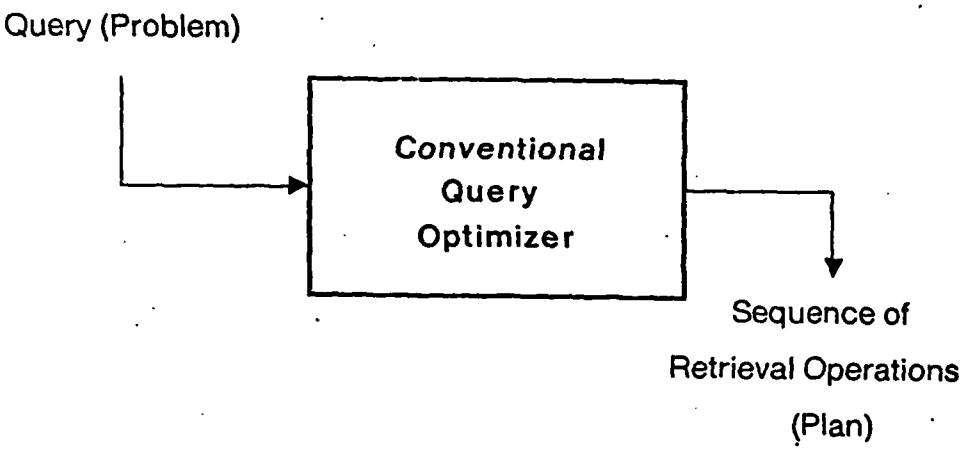


Figure 1-1: Operation of a conventional query optimizer

In operational terms (Figure 1-2), QUIST encompasses:

- a problem reformulator
- a conventional query optimizer
- a query selector.

QUIST starts with the constraints specified in the input query. The problem reformulator (Figure 1-3) first determines which database relations, if any, are *constraint targets.* QUIST designates a relation as a constraint target if it determines that it may lower the cost of query processing by finding additional contraints on that relation. If there are no targets, QUIST merely returns the original query for processing. Otherwise, its problem reformulator next repeats a cycle of operations that produce constraints until no more cycles can be carried out. During each cycle, relevant semantic integrity rules are retrieved. QUIST filters the rules according to the list of constraint targets, tests them for applicability against the current constraints, and asserts new constraints if possible. The process terminates when some cycle fails to generate new constraints. Finally, the problem reformulator groups the known constraints, both those given originally in the query and those derived using semantic knowledge of the domain, into a set of queries that are semantically equivalent to the original query.

QUIST next uses its conventional query optimizer to estimate the cost of processing each of the semantically equivalent queries. Finally, as its output QUIST selects the query with the lowest estimated processing cost as determined by the conventional query optimizer.

In more abstract terms, QUIST operates at three levels that correspond to the levels of the *plan-generate-test* paradigm of artificial intelligence [Feigenbaum71] seen in such systems as Meta-Dendral [Buchanan76]. The planning and generating steps take place in the problem reformulator, while the testing step is carried out by the conventional optimizer and the query selector.

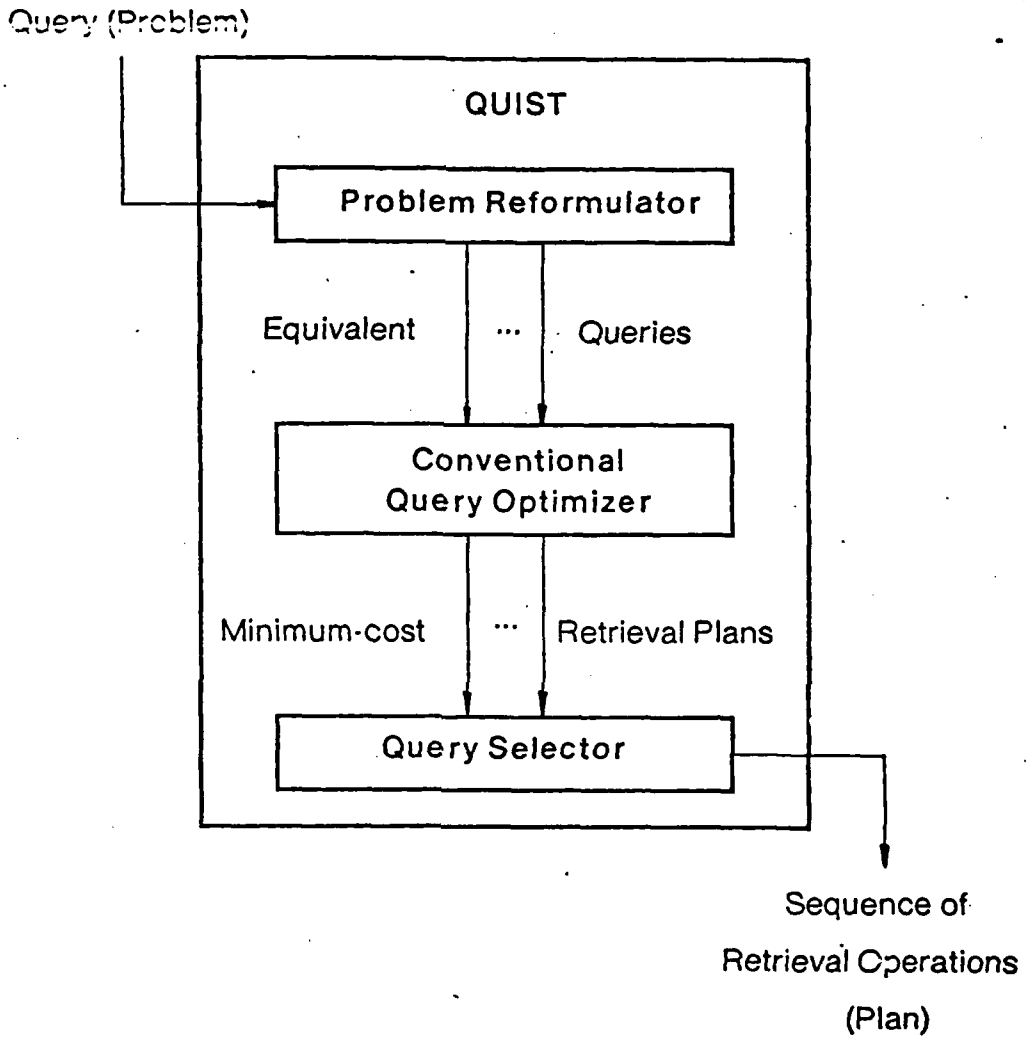


Figure 1-2: Operation of the QUIST semantic query optimizer

- The planning level is the one at which constraint targets are established. At this level, the query is treated in very abstract terms. The query's only important characteristics are the names of the relation attributes that are constrained or from which output values are requested. At the planning level, QUIST divides the database's relations into those that should be targets for inference and those that should not, much in the way that the Dendral program [Feigenbaum71] divides fragments of chemical structures into those that should or should not be part of a desired complete structure.

- At the generate level, QUIST explores a space of semantically equivalent queries. Each move in this space is a query transformation based upon the inference of an additional constraint. Each inference is supported by a rule in the semantic knowledge base. Only

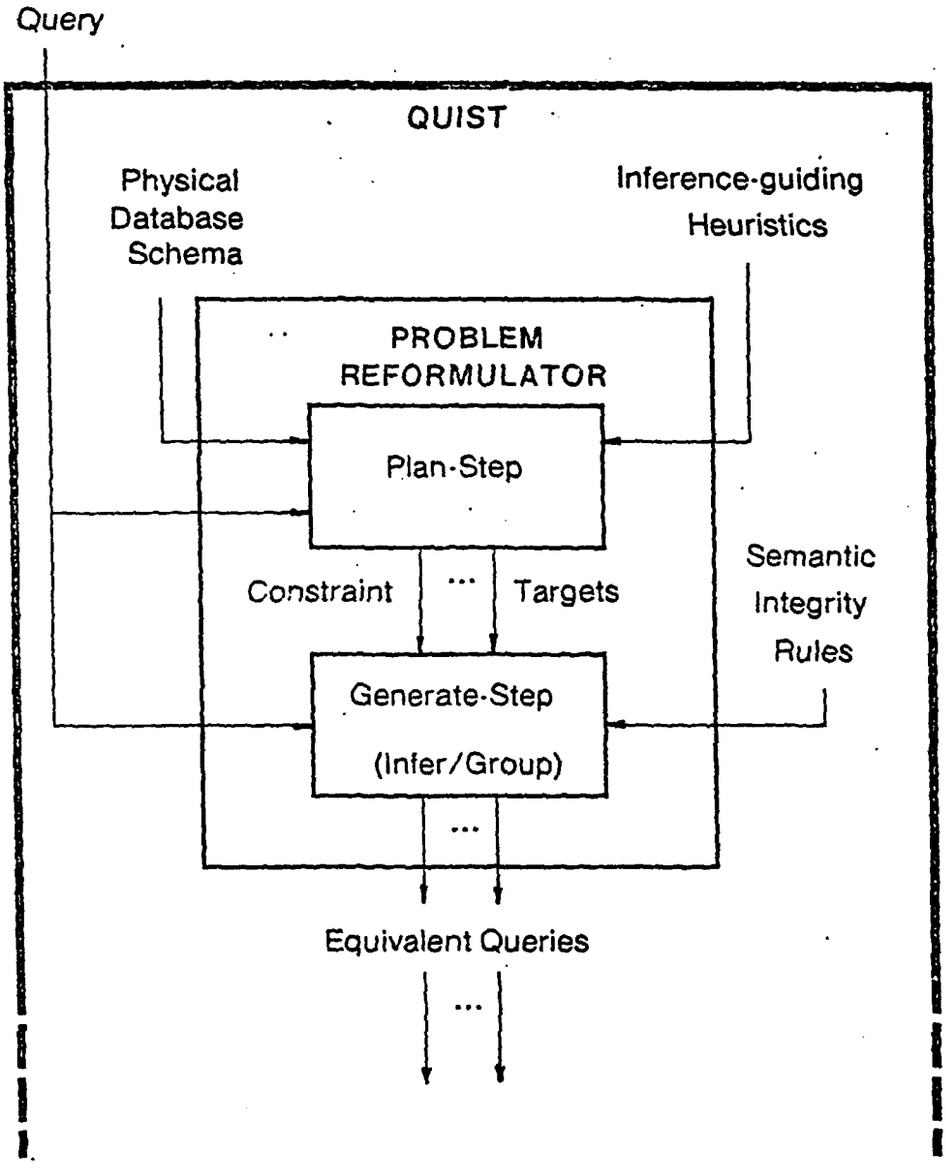


Figure 1-3: The QUIST problem reformulator

plausible moves are generated in the sense that the constraint target list permits only transformations which may possibly produce a lower cost query. The representation of the query at the generate level is less abstract than at the planning level. At the generate level, it is necessary to deal with the precise constraints on database attributes, not just with the names of the attributes that have been constrained.

- The testing level views each query in the most detail. Here, estimates of actual processing cost are obtained. QUIST includes a conventional query optimizer to find the least expensive way to process each semantically equivalent query produced at the generate level. The sequence in which each part of the query is processed is an essential factor. Processing at this level can be regarded as carrying out moves in a space of physical realizations of a single logically expressed query produced at the level above.

QUIST's use of the query processing expertise developed in conventional query optimization research is seen in the relationship between the searches at the generate and testing levels. The testing level search of alternative processing sequences, which is nothing other than conventional query optimization, is guided by detailed models of the cost of data access. The generate level search of semantically equivalent queries is guided by the constraint target list. The heuristics that produce the constraint target are, in effect, summaries or abstract versions of the detailed cost models used at the testing level.

1.2 Background of the current research

This research introduces the use of semantic reasoning to address the problem of query optimization in relational databases. In this section, we briefly review the research on database abstraction that has focussed attention on the query optimization problem. We indicate why relational databases are a suitable context for this current study, and we note how previous investigators have defined the problem in that context. We also discuss the important ideas about the semantic integrity of databases that suggest the possibility of semantic reasoning as an approach to efficient query processing. The research discussed in this section serves to frame the issues of the current study. We defer until Chapter 6 a discussion of the significant contributions of our research in the context of previous investigations.

1.2.1 Database abstraction and data models

The query optimization problem arises from the distinction between the logical and physical representations of a database. Fry and Sibley [Fry76] trace the evolution of database abstraction concepts that has led to the current notion of a *data model* as the means to maintain the logical/physical distinction. A data model is a language in which to express the logical structure of a database and the logical operations that are permitted upon that structure. That is, a data model is a vehicle for defining a database's data elements, relationships, and data types, as well as the operations

on the elements and relationships. It is also the basis for the *query language* by which elements of the database can be specified and retrieved by their logical properties.

Most attention has been paid to three kinds of data models. These are the network model [Taylor76], the hierarchical model [Tsichritzis76], and the relational model [Codd70]. Numerous database systems and query languages have been based on each of these models (see [Wiederhold77] for an extensive catalogue of such systems).

1.2.2 The relational model

Whatever the merits of database systems based on the three major data models with respect to ease of implementation, maintenance, and use, it has been persuasively argued [Date77] that the relational model provides the greatest separation between logical and physical levels of a database. The degree of data independence thus offered has stimulated much research into high level nonprocedural facilities for retrieval and update, for the definition of logical and physical structures and user views, and for the control of access, integrity, concurrency and recovery ([Kim79]); the current study follows in that line of research.

For simplicity, a relational database can be viewed as a collection of tables of data. In this view the table columns are attributes and the rows correspond to individual data records. There are no explicit connections among the tables, so manipulations of them can be specified simply and flexibly. One broad class of relational data manipulation languages is based on the relational algebra [Codd70] which defines operators to transform tables into other tables. Basic operators include restriction (horizontal subsetting of a table), projection (vertical subsetting of a table), and join (cross matching of two tables). Another broad class of languages is based on the relational calculus [Codd71], an applied predicate calculus.

1.2.3 Conventional query optimization

Research in conventional query optimization is important to the present work for two reasons. First, it has shown that separating the logical and physical aspects of a database does not necessarily result in inefficient query processing. Secondly, it has established a body of knowledge about the factors that govern the cost of processing queries, knowledge that can be directly applied in a system for semantic query optimization.

As we shall detail in Chapter 2, research in conventional query optimization has centered on queries built up from the basic relational algebra operators of restriction, projection, and join, or from their equivalents in the relational calculus.

The starting point for query optimization research in the relational context was the analysis of individual operations. Astrahan and Chamberlin [Astrahan75], among others, studied the restriction operation. Gotlieb [Gotlieb75] and Rothnie [Rothnie75] are among those who investigated the join operation.

Query optimization research builds on studies of the simplest queries that involve all of the key relational operations. These simple queries involve a single join operation between just two relations. The most important of these studies, from the standpoint of establishing the necessary elements underlying conventional query optimization, are those of Blasgen and Eswaren [Blasgen77] and Yao and DeJong [Yao78]. These studies produced sets of query processing methods for the simple queries, along with cost formulas and applicability conditions for the methods.

Most recently has come the development of optimizers for the general class of restriction-join-projection queries. The building blocks of these optimizers are the methods to process the single-join queries. The key insight underlying the optimizers is to see the evaluation of the general query in terms of a sequence of evaluations of simple queries. The task of the general optimizer is to choose which simple methods to apply, and in what sequence. The query optimizers of INGRES [Youssefi78] and of System R [Selinger79] can be viewed as operating in this manner.

1.2.4 Semantic integrity of databases

The idea of semantic query optimization presented in this thesis rests squarely on the concept that a database should be an accurate reflection of some real world application, not just any collection of data values. If the database contains values that cannot be attained in its real world application, then there is said to be a violation of the *semantic integrity* of the database. Semantic query optimization relies on a knowledge base of rules that are not part of the database proper, but that describe what values in the database correspond to possible states of the real world application.

In early database systems, integrity checks were confined to the detection of errors in format or were implemented as ad-hoc procedures incorporated in general database updating routines. A more systematic approach to the classification, detection and treatment of semantic integrity violations arose in the work of such researchers as Eswaren and Chamberlin [Eswaren75] and Hammer and McLeod [Hammer75]. Two broad notions of semantic integrity have been developed. One notion concerns the specification of permissible *states* of the database. For instance, it may be required that the salary of employees be no greater than some maximum figure; any data value for salary that exceeds that maximum does not reflect a legitimate state of affairs in the company, and so must be considered a semantic integrity violation. The other notion of semantic integrity concerns permissible *transitions* from one state to another. For instance, it may not be permissible to reduce the salary of an employee, even if the salaries before and after the change are both legitimate salaries.

Because query processing is assumed to take place in a single state of the database, we are only concerned in the present research with semantically based constraints on states of the database, rather than with permissible transitions between states. Several methods have been suggested for expressing such state constraints including: as qualifications in a query language expression [Stonebraker75]; in a special constraint language [Mcleod76]; in terms of an algebra in the sprit of abstract data types [Brodie78]; and in a general logical formalism such as predicate calculus [Chang78] or semantic networks [Roussopoulos77]. The research presented here generally adopts the predicate calculus

approach to representing semantic integrity constraints adopted by Chang [Chang78] and others [Gallaire78].

In addition to studies in the representation of semantic constraints on databases, much effort has been devoted to issues of designing systems for specifying semantic integrity constraints and for checking them efficiently ([Hammer78], [Wilson80]). The method devised by Stonebraker [Stonebraker75] for maintaining semantic integrity of a database is similar to the method used by QUIST. Stonebraker's method, called *query modification*, works by modifying an update request. In general terms, it conjoins appropriate integrity constraints to the qualification portion of the update request. In this way, no data is altered that would result in a state that violates the conjoined constraint. The query transformations described in the present research are similar to these query modifications.

We have now concluded our brief review of research that forms the background to our investigation of semantic query optimization. We shall look at additional related research when we discuss the significance of our results in Chapter 6.

1.3 Guide to reading

Semantic query optimization integrates two important sources of knowledge: knowledge about cost factors in query processing, and knowledge about the semantics of the application task domain. Chapter 2 discusses the problem definitions and models of query processing that characterize conventional approaches to the optimization of queries in relational databases. Chapter 3 introduces semantic query optimization. It presents the formal basis for the notion of a transformation of a relational query that preserves meaning in all permitted states of the database.

In Chapter 4, we describe the QUIST system in detail. We show how the models of query processing developed in research on relational databases are directly incorporated into heuristics to guide the transformation of queries into less costly, semantically equivalent forms. In Chapter 5, we discuss the effectiveness of QUIST in terms of the estimated reductions in cost made possible by various kinds of query transformations. We also report the results of using QUIST on a range of queries that illustrate those classes of transformations, and we discuss the stability of the QUIST control strategy when the size of the database or rule base increases.

Finally, in Chapter 6 we discuss the significance of the work reported here in the context of research in database management and artificial intelligence. We also review the limitations of the current work and suggest directions for future research.

Chapter 2
Query Processing Expertise

The most important outcome of conventional query optimization research† is deeper understanding of the problem of accessing data, what we might call *query processing expertise.* In the terminology of Section 1.1, query processing expertise is knowledge of problem-solving methods for the specific problem of processing database queries. This expertise is manifested in two ways: in terms of the assumptions, models, and approaches that characterize advanced query optimization systems, and in terms of generalizations concerning the factors that contribute to the cost of processing queries.

Research in the optimization of relational database queries serves as an appropriate case study for examining query processing expertise. There is substantial agreement on the validity and power of the data storage, data access, and cost models developed in the relational context, although there is no standard accepted for all systems.

Query processing expertise is an essential underpinning of semantic query optimization. Through the proper use of this knowledge, it is possible to control the use of semantic knowledge in an effective semantic query optimization system. In this chapter, therefore, it is our aim to summarize the expertise that has emerged from research on relational database query processing. In so doing, we specify the class of queries towards which we have directed our specific research in semantic query optimization.

In Section 2.1, we review the basic terminology of relational databases. In Section 2.2, we describe how queries are specified. Our objective in Section 2.3 is to specify the knowledge that underlies conventional approaches to optimization of restrict-join-project queries. This is the class of queries that is the focus of conventional query optimization research. We accomplish this objective through a detailed review of some characteristic research work in the field. We extend this in Section 2.4 to show how this query processing knowledge is actually used in a conventional query optimizer. Finally, in Section 2.5 we make explicit some of the generalizations about query processing that constitute query processing expertise and that play an integral part of semantic query optimization.

---

†The use of the term *optimization* in this context is discussed in Section 1.1.

2.1 Relational databases

In formal terms, a *relational database* is a collection of *relations*. Let D1,D2,. .,Dn be n sets. Codd [Codd71] defines the *extended Cartesian product* of these sets as:

$$X(D1,D2,...,Dn) = \{(d1,d2,...,dn): dj \in Dj \text{ for } j = 1,2,...,n\}.$$

R is a relation on the sets D1,D2,...,Dn if it is a subset of the extended Cartesian product of those sets. A relation is therefore a set, and its members are n-tuples (or more simply, *tuples*) where n is referred to as the *degree* of the relation. The sets Di on which the relation R is defined are called its underlying *domains*. For purposes of modelling databases, the domains under consideration are integers and character strings. The number of n-tuples (or more simply, tuples) in R is the *cardinality* of R.

A relation can be viewed as a table in which the rows correspond to tuples and the columns correspond to mappings from the relations into its domains. The mappings are called *attributes*, and it is possible to base more than one attribute on the same domain. An attribute value is the entry for a particular row/column combination.

To illustrate the data definitions, consider a relation that describes characteristics of ships:

SHIPS(Ship Owner Type Length Draft Deadweight)

SHIPS is the name of a relation. The words in parentheses are the names of the attributes of the relation.

Assume that the attributes Ship, Owner, and Type are defined on the strings, and that the other attributes take on integer values. The relation might consist of the following tuples (based on data from Lloyd's Register of Ships [Lloyds78]):

Ship	Owner	Type	Length	Draft	Deadweight
"Bralanta"	"Braathan"	"Tanker"	285	17	154
"British Wye"	"BP Shipping"	"Tanker"	171	9	25
"Carlova"	"Index Maritime"	"Bulk"	218	12	55
"George F. Getty"	"Hemisphere"	"Tanker"	319	19	227
"Intellect Energy"	"Energy Shipping"	"Tanker"	88	6	2

Figure 2-1: Illustrative tuples in the SHIPS relation

We take the first tuple to signify that the tanker Bralanta is owned by Braathan, that it is 285 meters long, has a draft of 17 meters, and has a deadweight (size) of 154,000 long tons. The other tuples are interpreted similarly.

2.2 The specification of relational database queries

There are two broad classes of relational query languages: those based upon the *relational algebra* and those based upon the *relational calculus*. An extensive discussion and comparison of the two classes can be found in [Date77]. In a relational algebra language, a query is expressed by specifying operators that transform relations into other relations, and ultimately into the desired result relation. The relational calculus is an applied predicate calculus. Query languages based on the relational calculus specify retrieval in terms of a calculus expression. Pirotte [Pirotte78] gives an excellent survey of the kinds of relational query languages that are based on the predicate calculus.

The languages based upon the relational algebra and the languages based upon the relational calculus present different interfaces to a user or a program. However, Codd demonstrated [Codd71] that the two formalisms are equivalent. In that same paper, Codd proposed the relational calculus as a standard against which the expressive power of query languages could be measured.

We now illustrate the specification of relational queries using a language based on the relational calculus, the SODA language [Moore79]. We note that a query does three things:

1. It specifies what relations are involved in the query, either for checking conditions or for retrieving specific values.

2. It specifies what conditions must be met.

3. It specifies what aspects of the qualifying data items are to be retrieved.

Our illustration uses a simple example relational database that includes two relations:

SHIPS(Ship, Length)

CARGOES(Ship, Cargotype, Quantity)

These relations contain information about ships and the cargoes they carry. Suppose it is desired to retrieve the names of ships longer than 200 meters that are carrying more than 1000 tons of wheat. The appropriate query in SODA could be expressed as:

```
(IN V1 SHIPS) (IN V2 CARGOES)

((V1 Ship) = (V2 Ship))

((V1 Length) > 200) ((V2 Cargotype) = "Wheat") ((V2 Quantity) > 1000)

(? (V1 Ship))
```

The first line defines the ranges of two *tuple variables* V1 and V2. A tuple variable is a variable that ranges over the tuples of a specified relation. For each tuple variable, the query specifies the relation over which the variable ranges. Thus, V1 ranges over the SHIPS relation, and V2 ranges over the CARGOES relation.

The next two lines of the query specify the retrieval qualification. What are the objects on which the qualification is to be tested? They are precisely the tuples of the relation formed by taking the Cartesian product of the relations over which the tuple variables range. The formation of the Cartesian product is conceptual. That is, it may not actually be necessary to form the product completely before applying qualifying conditions. Indeed, it is advisable on efficiency grounds not to form the product. Each tuple in the Cartesian product of the example query is the concatenation of a SHIPS tuple with an CARGOES tuple.

The first line of the qualification contains the *join term* ((V1 Ship) = (V2 Ship)). A join term has the form (X COMP Y) where X and Y are *attribute specifiers*, and COMP is one of the comparison operators such as =, ≥, and so forth. An attribute specifier is a (tuple variable, attribute name) pair; it is the same thing as the *indexed tuple* referred to in [Codd71], but restricted to a single relation and a single attribute. The attributes specified by X and Y must be defined on the same underlying domain. Roughly speaking, the join terms of our example query pairs each cargo with the ship that is carrying it by equating the names of ships.

The next line contains three *restriction terms*, ((V1 Length) > 200), ((V2 Cargotype) = "Wheat"), and ((V2 Quantity) > 1000), that further restrict the subset of the Cartesian product that passes the join term test. A restriction term is of the form (X COMP CONSTANT) where X and COMP are as before and CONSTANT is a constant in the domain of the attribute specified by X. In our example, the SHIPS portion of qualifying Cartesian product tuples must have a Length value greater than 200, and the CARGOES portion must have a Cargotype value of "Wheat" and a Quantity value greater than 1000. Note that there is an implicit conjunction among the join and restriction terms.

The tuples of the specified Cartesian product relation that satisfy the retrieval qualification are called the *qualifying tuples*. The final task of the query is to say what information is sought from the qualifying tuples. The desired output is specified in a *target list*. A target list is a list of attribute specifiers. Each attribute specifier in the target list requests the retrieval of the value of the specified attribute for all qualifying tuples. The final line of the example query specifies the retrieval of the ship name Ship from the SHIPS portion of the qualifying tuples.

In a standard relational calculus language, a retrieval qualification is a logical combination of restriction terms and join terms, using the standard logical connectives and existential and universal quantification. The reader is referred to [Codd71] or [Pirotte78] for a thorough discussion of allowable qualifications. Intuitively, the join terms of a qualification most often correspond to semantic relationships among entities and the restriction terms are additional restrictions on the entities so related.

2.3 Optimization of simple restrict-join-project queries

Experience with relational database systems indicates that there is a subset of the relational algebra in which a very great percentage of queries can be expressed. This subset has therefore become the main focus of conventional query optimization research in relational databases.

In this section, we look in detail at a characteristic example of this research, the work of Blasgen and Eswaren [Blasgen77] at IBM. Our purpose is to reveal the foundation elements of conventional query optimization that have been generally accepted by investigators in the field. These elements are noted in Figure 2-2.

1. A limited but important class of queries.
2. A model of data storage.
3. A model of access to data.
4. A cost measure related to data access.
5. A set of methods to carry out the "atomic" queries.
6. System and query parameters that are used in cost analysis.
7. Cost formulas and applicability conditions for the methods.

Figure 2-2: Elements of conventional query optimization

The queries under consideration are those that can be expressed in terms of the three basic relational algebra operations: *restriction*, which selects rows from a relation; *projection*, which selects columns from a relation; and *join*, which matches (cross-references) two relations on compatible attributes (attributes defined on the same underlying domain of values). Note that the discussion applies to the relational calculus too because a corresponding class of queries in that formalism can be translated into these algebraic terms (see [Yao79], for example). The corresponding class in relational calculus terms involves range statements for tuple variables, plus a qualification in terms of those variables, which together correspond to restriction and join, and a target clause which corresponds to a projection.

The work reported in [Blasgen77] considers a general query with the following form: apply a given restriction to relation R, yielding R', and apply a possibly different restriction to relation S, yielding S'. Join R' and S' to form a (new) relation T, and project some columns from T. This query can be termed a *two-relation query*, and can be viewed as the atomic unit from which all the restrict-join-project queries can be constructed.

Blasgen and Eswaren propose straightforward models of access and storage. Because these models are typical of the access and storage models used throughout conventional query optimization research, we will describe them in detail.

The database is assumed to be stored on direct access secondary storage, typically on disk. Physical storage is divided into fixed length pages. There are two kinds of pages, *data pages* and *index pages*. The tuples of the database relations are stored as fixed length records on the data pages (under this assumption, the terms *tuple* and *record* are used interchangeably throughout the query optimization literature). A data page may contain tuples from more than one relation, but no tuple is broken up across page boundaries. Each tuple has a unique *tuple identifier* (TID). It is assumed that the file system can convert a TID into an address for direct access to a tuple. The TID's have the property that accessing a set of tuples in sorted TID order accesses a data page at most once.

Secondary storage is divided into *segments*. A segment is a large address space that contains one or more relations. It is implemented as a set of pages. Each tuple stored in a segment identifies the relation to which it belongs. No relation is broken up across segment boundaries. To obtain all the tuples of a relation, the segment can be scanned by fetching its pages one at a time and checking every tuple on the page for membership in the desired relation. This kind of scan is called a *segment scan*. A segment scan fetches every page in the segment once.

Because a segment can be large, a segment scan can be very slow. For this reason, other *access paths* to the tuples of a relation may be arranged. The model described in [Blasgen77] admits an access path based on a single-column *index* to a relation (another type of access path is the *link*). A single-column index on a column A of a relation R is a set of pairs whose first component is a value from A and whose second component is the TID of a tuple of R that has that value. The index is stored as a B-tree of pages. Pages at the lowest level contain the actual (key, TID) pairs sorted by key. Higher levels contain pointers to lower level pages. Because of the B-tree organization, the index permits rapid access to a single tuple with a desired value. The number of index pages to be fetched equals the height of the tree. Also, the lowest level index pages are linked so that all the tuples or any key subsequence of them can readily be retrieved in sorted key order by scanning the leaf nodes of the index. This operation is called an *index scan*.

The usefulness of an index for query evaluation depends upon whether the relation is *clustered* with respect to the index and on the number of tuples to be retrieved. A relation is clustered with respect to an index if the tuples of the relation are stored in the same sequence as the key sequence given by the index. With such an arrangement, if the index is used to access tuples of the relation then each data page of the relation will be fetched only once. On the other hand, if the index is

unclustered with respect to the relation, it is assumed that each access of a tuple using the index requires fetching a new data page.

Besides segment scan and index scan, the access model includes sorting tuples on the value of some column. It is assumed that the files are large enough to require an external Z-way sort-merge.

Based on the access and storage models, Blasgen and Eswaren develop methods to carry out the two-relation query. The methods differ in their use of TIDs, indexes, and sorting, and some of them are only applicable if certain indexes exist. Two of the methods will be described.

The first method is the *nested loops* method. The first relation is scanned. For each tuple that meets the restrictions on that relation, a scan of the second relation is performed. Some tuples from the second relation may be found that meet the restrictions on the second relation and that have the same join column value as the current first relation tuple. Each qualifying second relation tuple is combined with the current first relation tuple to form a composite result tuple (projecting out the desired columns).

The second method is the *merging scans* method. Both relations must be scanned in join column order. Either relation that is not indexed on its join column must be sorted into a temporary file that is ordered on that column. The first relation is scanned in join column order. For each first relation tuple that meets the restrictions, the second relation is scanned. However, because of join order sequencing, it is possible to keep track of the current position of the two scans and never rescan any portion of either relation once the current join column value exceeds the value in that portion. This bookkeeping also makes it possible to spot situations where tuples in one relation have no join partners in the other relation.

The *cost measure* for the methods is the number of pages that must be brought in from secondary storage. This is a reasonable assumption if it is believed that input/output time dominates processor time. Most (though not all) query optimization models make this assumption.

Given a set of methods and a cost measure, it is possible to develop cost formulas for the methods. The formulas depend upon system parameters that are *database-dependent* but independent of the specific query, and upon other *query-dependent* parameters. The cost formulas for a method to process a complete two-relation query are built from cost formulas for scanning a single relation. For a segment scan, the cost is the number of pages in the segment that contains the relation. This is obviously a system parameter, not related to the restrictions or other aspects of the query.

Unlike a segment scan, the cost of an index scan depends both on system parameters and on query parameters. To see this, consider a scan of a key subsequence of column A of relation R using index I. Suppose the scan starts at column value V1 and ends at column value V2. That is, the aim is to retrieve all tuples of relation R that have a value for column A that is greater than or equal to V1 and less than or equal to V2.

The first step of the scan is to locate the first tuple with a value between V1 and V2. The index permits rapid access to that tuple, at the cost of fetching a number of index pages equal to the height

of the tree in which the index is stored; index tree height is clearly a system parameter. The rest of the operation consists of chaining along the leaf index pages until the key value exceeds V2. At each index page, TIDs for qualifying tuples are found and their data pages are fetched. The number of leaf index pages and the number of data pages fetched depends upon the number of tuples that have a value between V1 and V2 for column A. This obviously depends upon the query because V1 and V2 are specified by the query.

It is straightforward to combine scan cost formulas into cost formulas for a complete two-relation query processing method. For example, the nested loops method consists of a scan of one relation and for each of its qualifying tuples, a scan of the second relation. Hence, the cost of the nested loops method is the cost of scanning the first relation plus the product of the number of qualifying first relation tuples with the cost of scanning the second relation.

The work of S.B. Yao and his associates presented in [Yao78] and [Yao79] rests on the same elements as the work of Blasgen and Eswaren. In particular, Yao's work addresses the same class of two-relation restrict-join-project queries and presents similar storage, access, and cost models. The work is significant in systematically building the query processing methods out of a comprehensive set of submethods. This results in a much larger set of query processing methods than Blasgen and Eswaren present. Yao also investigates the use of links as auxiliary access paths.

2.4 A conventional query optimizer for multifile queries

The methods that have been developed to handle two-relation queries in the restrict-join-project class have been extended to handle queries that involve n relations, where n is greater than two. This is the basis for the general query optimizer for IBM's System R experimental relational database management system [Selinger79]. The optimization methods for the INGRES relational database system can also be viewed in this framework for most queries [Youseffi78]. We illustrate the functioning of n-relation conventional optimizers with the System R optimizer. The discussion omits some aspects of optimization that are specific to System R, such as the possible requirement to present results in a specified sequence or grouping.

Processing a query that involves N relations is viewed as processing a sequence of queries that involve two relations. In this view, a two-relation subquery is processed to form a resulting composite relation. This relation is processed with a third relation to form a new composite, and the sequence continues until all relations in the original query have been brought in. In the actual processing, it is not always necessary to form and store the complete composite relation before the next relation is brought in. Instead, when a composite tuple is formed from a two-relation query, it can be joined with tuples from a third relation, and so forth. Intermediate composite relations are stored only if a sorting operation is required in connection with the next two-relation processing step.

The extension from two-relation queries to N-relation queries outlined above has been termed *iterative composition* by Kim [Kim79]. The task of the general query optimizer based upon iterative

composition of subqueries is twofold. First, it is necessary for the optimizer to choose the order in which the relations are to be brought in; that is, it must choose the sequence of two-relation subqueries. Second, the optimizer must choose a method to carry out each subquery.

The sequence of subqueries is important in determining the overall cost, even though the size of the result is the same regardless of the processing sequence. For a query that involves N relations, there are N factorial permutations of the processing sequence. However, the method to bring in the K+1th relation is independent of the way the first K relations are processed. The search of sequences can therefore operate efficiently by finding good sequences for successively larger subsets of the relations in the query. The System R optimizer uses another heuristic to reduce the number of permutations it considers: a relation is considered as the next one to be brought in only if it is involved in a join with one that has already been processed.

The System R optimizer grows a processing search tree by iteration on the number of relations involved so far. First, the best method is found to scan each relation. Next, the best way is found to involve the first relation in a two-relation query with a second relation. This continues until all relations are involved. Unpromising paths of the search tree are pruned on the basis of the heuristics described above and on the basis of estimated processing costs for the partially worked out queries.

An important source of information for the optimizer is the estimated *selectivity* of the query restrictions, the only place in the optimizer where semantic information is used. The selectivity of a restriction on a relation is the fraction of tuples of the relation that satisfy the restriction. Both the cost formulas for certain scans and the formulas for combined methods use the fraction of tuples that meet the restrictions imposed by the query. To estimate selectivity, the optimizer uses information about the range of values for attributes, if that information is available. It makes the simple assumption that the values for any attribute are uniformly distributed within the legal range and that the distribution of values is known with sufficiently fine granularity. This assumption enables estimates to be made with limited statistics on database values. Youseffi [Youseffi78] has looked into the issue of how additional statistics can improve the estimates, but the simple System R methods appear to work fairly well [Astrahan80a]. In the absence of value range statistics, the System R optimizer makes arbitrary although intuitively plausible estimates.

2.5 Generalizations about query processing

In this section, we review some general conclusions about relational query processing that can be drawn from the kind of research described above. As we shall see, these general conclusions play an important role in the design of an effective semantic query optimization system.

The net result of conventional query optimization research is an appreciation of how the relationship between the constraints specified by a query and the data structures comprising the database affect the cost of processing. In many cases, this knowledge is represented in the choice of relevant system parameters and in the cost formulas based upon them. Occasionally, though, the knowledge is expressed as general statements about the interaction of queries and structures.

A key factor in the cost of processing a query is the physical clustering of logically related items, what Wiederhold [Wiederhold77] refers to as the *binding* of the data semantics. While this seems intuitively obvious, the studies of Blasgen and Eswaren demonstrate the degree of its importance, and they relate it to specific kinds of queries and specific structures in the storage model. The role of clustered indexes is highlighted. As the System R experiments [Astrahan80a] confirm, the case of an equality predicate on an indexed but nonclustered attribute is about the only case in which a nonclustered index scan is preferred over a clustered one. In general, a query whose constraints permit the exploitation of clustered access paths, whether indexes or links, can be answered much more efficiently than a query whose constraints do not permit those paths to be exploited.

Conventional query optimization pays attention to avoiding catastrophically bad processing methods. The classic example of a bad processing method is processing a join as a Cartesian product followed by a restriction. In one of the rare glimpses into the explicit reasoning of experts in query processing, Youseffi and Wong [Youssefi79] discuss the formulation of processing strategies based on this consideration. They note that, intuitively, the processing cost for a one-variable query is linear in the size of the relation, while the cost for a two-variable query increases faster than linearly in the sum of the relation sizes. This line of reasoning suggests to them that it is nearly always advantageous to restrict the individual relations prior to checking the join condition, that is, prior to accessing and considering the relations together. An exception occurs when one of the relations is physically clustered with respect to the join condition. Other factors to be considered are the sizes of the relations and whether the relations are in the target list. In any event, it is generally true that the stronger the restrictions that can be applied to the individual relations prior to carrying out the join, the less expensive is the overall process.

This discussion is indicative of the body of expertise about query processing that has emerged from research on query optimization. To restate the main idea, the cost of processing depends on the relationship between the constraints specified by a query and the data structures implicated by the query. Specifically, with respect to the fundamental operations discussed in this chapter, Figure 2-3 indicates some representative generalizations:

In Chapter 4, we shall see how such generalizations are used to control the way semantic knowledge is used in semantic query optimization. Before that, however, Chapter 3 discusses the shortcomings of conventional query optimization and describes the new approach of semantic query optimization.

- G1. A restriction on an attribute that is not indexed leads to an expensive scan.
- G2. A restriction (other than an equality predicate) on an indexed attribute where the index is not a physically clustering index leads to an expensive scan.
- G3. A restriction on a physically clustering index can be processed efficiently.
- G4. The cost of joins generally dominates the overall cost of processing.
- G5. A join between two large and weakly restricted relations is very expensive.
- G6. The cost of a join decreases substantially as the strength of restrictions on the joined relations increases, except on a relation which is clustered with respect to the join term (and is therefore likely to be the "inner" relation of the join method).

Figure 2-3: Generalizations about query processing

Chapter 3
Semantic Query Optimization

In this chapter, we present the formal basis for semantic query optimization in relational databases. We begin in Section 3.1 by reviewing the limitations of conventional query optimization that motivate the development of our new method. In Section 3.2 we look informally at the notion of the *semantic equivalence* of two database queries that is at the heart of semantic query optimization. Semantic equivalence is defined in terms of semantically meaningful states of the database. This in turn is intimately bound up with with the *semantic integrity constraints* associated with the database. We formally define semantic integrity constraints for relational databases in Section 3.3. In Section 3.4, we show informally how one query can be transformed into a semantically equivalent one using a semantic integrity constraint. Section 3.5 synthesizes the preceding sections into a formal definition of relational database query transformations that preserve semantic equivalence. Finally, Section 3.6 discusses additional logical equivalence transformations that can be used in conjunction with semantic equivalence transformations to reduce the cost of processing a query.

3.1 The limits of conventional query optimization

Conventional query optimization research has identified a set of problems, has produced useful models of data storage and file operations, has yielded insights into the factors that influence the cost of query processing, and has in general lent support to the belief that high level query languages can be used with acceptable efficiency.

The difficulty with conventional query optimization remains the lack of correspondence between the logical relationships referenced in a query and the physical relationships of the data that represent them. One can view the manipulations of a conventional query optimizer as a *hunt for opportunities*, for those parts of the query in which the logical structure corresponds well to the supporting physical structure. For instance, the presence of indexes on the joining attributes for two files in a multifile query is likely to make that join a candidate for processing before other joins. The logical/physical correspondences are exploited to reduce as much as possible the size of the data structures that must be handled without suitable physical support.

To maintain physical support for all logical relationships is not possible, however. The costs to maintain that support as the database evolves are too great. In simplest terms, if a query involves a

large amount of data in logical relationships that are not well structured in the physical database, then the query can't be processed efficiently. Examples of poor correspondence are easy to imagine. A constraint on an unindexed attribute of a relation forces a sequential scan. A join between two files for which suitable indexes or links do not exist forces a cross matching process which is almost always very expensive.

Conventional query optimization is limited to treating the logical restrictions of the query as fixed. If the query restrictions cannot be processed efficiently, nothing can be done.

3.2 The semantic equivalence of queries

We now begin the formal description of *semantic query optimization*, developed as a response to the limitations we have just described. The key idea is that the given query restrictions are not regarded as fixed, but as perhaps only one of several equivalent ways to pose the same question. We said in Section 3.1 that conventional query optimization is a hunt for opportunities. The goal of semantic query optimization is to create new search spaces in which to hunt for such opportunities.

The heart of semantic query optimization is the process of transforming a query into a *semantically equivalent* one. Two queries are considered to be semantically equivalent if they result in the same answer in any state of the database that conforms to the semantic integrity constraints (see Section 1.2.4).

Semantic equivalence is not the same as *logical equivalence*. Two queries are logically equivalent if the qualifications of one can be transformed into the qualifications of the other by the application of standard logical equivalences such as De Morgan's Laws. Another way to put this is that two queries are logically equivalent if they produce the same answer in any database whatsoever in which the queries are well-defined. For instance, the query "list the names of all employees who are not both unmarried and over forty years old" is logically equivalent to the query "list the names of all employees who are either married or are not over forty years old."

Logically equivalent queries are obviously semantically equivalent, but semantically equivalent queries need not be logically equivalent. That is, two semantically equivalent queries might yield different answers when posed to the database in a state where some semantic integrity constraint is violated.

For example, suppose there is a semantic integrity constraint to the effect that the company has no employee under the age of eighteen. If the database conforms to this condition, then the query "list the names of all employees between the ages of fifteen and twenty" is semantically equivalent to the query "list the names of all employees between the ages of eighteen and twenty." The answers will be the same because the enforcement of the semantic integrity constraint guarantees that there is no item in the database corresponding to an employee between the ages of fifteen and eighteen. However, if a violation of the constraint is permitted and data is entered on an employee whose age is recorded as sixteen years old, then the two queries will not produce the same answer.

Another way to look at the difference between logical equivalence and semantic equivalence is that semantic equivalence is measured against a particular set of semantic integrity rules. For instance, if the rule requiring employees to be at least eighteen is changed so that employees must be at least seventeen instead, then the two queries just discussed are no longer semantically equivalent. The first query may return some seventeen year olds but the second one cannot. By constrast, logical equivalence is unaffected by changes in the semantic integrity constraints.

We also wish to distinguish semantic equivalence from coincidental equivalence in a particular state of the database. Semantically equivalent queries must produce the same answer in all permitted states of the database. A simple example illustrates what we mean by coincidental equivalence. Suppose the company happens to have one employee named "Fred Smith" and that he happens to be the only employee who is 47 years old. Then the queries "What is the employee number of each employee named Fred Smith?" and "What is the employee number of each employee who is 47 years old?" give the same answer. However, it is easy to imagine a situation in which the two questions do not give the same answer. For instance, nothing prevents the company from hiring another 47 year old employee whose name is not "Fred Smith". If the company does hire another 47 year old, then the two questions do not have the same answer.

3.3 Semantic integrity constraints

The foregoing discussion of semantic equivalence underscores the point that:

> *The basis of semantic equivalence independent of logical equivalence and independent of changes in state is the enforcement of the semantic integrity of the database.*

The notion of the semantic integrity of a database is understood with respect to the relationship of the database to some real world application. Every allowable state of the database is supposed to be a valid "snapshot" of aspects of the application. If the database contains values that cannot be attained in the real world application, then there is said to be a *violation* of the semantic integrity of the database.

We now formally develop the notion of semantic integrity constraints for relational databases. In so doing, we are also preparing the groundwork for a formal discussion of relational database queries and semantic equivalence transformations.

Our point of view is a standard one in research analyzing databases in terms of formal logic (see, for instance, [Gallaire78]). The descriptors of relations and queries are just those of the relational calculus that we discussed in Section 2.2. A relational database is considered to be made up of two parts: an *extensional database* (EDB), and an *intensional database* (IDB).

The EDB is the set of elementary assertions or tuples contained in the relations in any particular state of the database. For instance, any of the tuples in our example in Section 2.1, such as

("Bralanta" "Braathan" "Tanker" 285 17 154)

is part of the EDB.

The IDB is a set of general laws expressed as closed well-formed formulas in the first-order predicate calculus. The general laws, as the name implies, apply more broadly than the elementary assertions. An example of a general law that applies to all the tuples in a single relation is a rule that all ships over 190 thousand tons deadweight (size) are supertankers. This can be expressed as

$$\forall x/_{SHIPS}(x.\text{Deadweight} > 190) \rightarrow (x.\text{Shiptype} = \text{"supertanker"}).$$

As noted above, this general rule is expressed as a closed well-formed formula in a typed first-order predicate calculus. The variable x ranges over tuples of the SHIPS relation. The expression "x.Deadweight" signifies the value for the Deadweight attribute of the tuple to which x is bound.

Other general laws may involve more than one relation. Suppose there is a CARGOES relation that includes Ship and Quantity attributes. Then we can express the rule that a ship cannot carry a greater quantity of cargo than the ship's capacity as follows:

$$\forall x/_{SHIPS}\, \forall y/_{CARGOES}\, (x.\text{Shipname} = y.\text{Ship}) \rightarrow (y.\text{Quantity} \leq x.\text{Capacity})$$

Intuitively, most general laws involve universal quantification over relations. However, it is also possible to express an existential law, such as the rule that there is at least one supertanker:

$$\exists x/_{SHIPS}\, (x.\text{Shiptype} = \text{"supertanker"}).$$

Why divide the database into extensional and intensional parts? The reason is the following essential relationship between EDB and IDB:

> *The elementary assertions or tuples of the EDB are considered to define an* **interpretation**† *of a first order theory whose proper (nonlogical) axioms are the general laws of the IDB.*

From the perspective of semantic query optimization, the importance of general laws stems from their use as integrity rules. In terms of a first-order theory and its interpretation, every operation on the database such as adding, deleting, or changing elementary assertion, amounts to a change in interpretation. In these terms, we have the following definition:

> *Semantic integrity is enforced if and only if the only changes permitted to the database are those that leave the elementary assertions of the EDB as a* **model** *(and not merely an interpretation) of the semantic integrity rules of the IDB.*

In other words, the enforcement of semantic integrity prevents the database from entering a state in which any of the closed well-formed formulas of the integrity rules evaluates to false.

---

†See [Nicolas78a].

3.4 Query transformations that preserve semantic equivalence

Our motivation in investigating semantic integrity constraints is to see how to transform a query into a semantically equivalent query. The significance of integrity rules for this purpose becomes apparent if we consider the notion of satisfiability. Specifically, suppose we drop the universal quantifier from the first general rule above. The result is an open formula in which the previously quantified variable now appears free. The open formula can be put in the form of a query, similar to the form that appears in [Pirotte78]:

$$Q_i: \{x/_{SHIPS} \mid (x.Deadweight > 190) \rightarrow (x.Shiptype = \text{"supertanker"})\}.$$

The answer to this query is a set of tuples from the SHIPS relation, namely the set of tuples that corresponds to ships whose deadweight is less than 190 or which are supertankers. For convenience, we omit any indication of which attributes of x should be returned.

The items in the answer set for this query are those tuples in the SHIPS relation which, when substituted for x in the formula, make the formula true.

The significant observation is that by enforcing the original integrity constraint, we require that every tuple in SHIPS make the formula true. Hence, the open formula is satisfied by the entire SHIPS relation. That is to say, according to the rule, *every* ship either has a deadweight of less than 190 or is a supertanker.

Consider any other query that requests the set of tuples from SHIPS that satisfy some qualification Q. Let T be the set of qualifying tuples. The set T is clearly a subset of the set of all tuples in SHIPS. But all tuples in SHIPS satisfy the integrity constraint qualification Q_i, so in particular, the tuples in T satisfy it also. That is, no tuple of T satisfies the qualification Q but does not satisfy the qualification Q_i. Therefore, we can replace qualification Q by the conjunction of Q and Q_i, and the answer set T remains the same. The query with this new qualification is semantically equivalent to the original query with qualification Q.

For example, suppose we start with a query that requests all ships with a draft greater than 20 meters. We express this as:

$$\{x/_{SHIPS} \mid (x.Draft > 20)\}.$$

We can drop the universal quantifier from the integrity constraint and obtain the following semantically equivalent query:

$$\{x/_{SHIPS} \mid (x.Draft > 20) \wedge ((x.Deadweight > 190) \rightarrow (x.Shiptype = \text{"supertanker"}))\}$$

This new query doesn't make much sense as it stands. It asks for those ships whose draft exceeds 20 meters and which, if they have a deadweight over 190, also have a shiptype of "supertanker". Nevertheless, this query yields the same answer as the original one. Now suppose we had started with a query that requests all ships with a deadweight of over 190 thousand tons:

$$\{x/_{SHIPS} \mid (x.Deadweight > 190)\}.$$

We apply the same transformation to this query to obtain:

$$\{x/_{SHIPS} \mid (x.Deadweight > 190) \wedge ((x.Deadweight > 190) \rightarrow (x.Shiptype = \text{"supertanker"}))\}$$

The transformation based upon integrity semantics has been completed, but we can now use the logical axioms of first-order logic to transform this query further. In particular, we use the equivalence expressed in the axiom schema:

$$(A \wedge (A \rightarrow B)) \equiv (A \wedge B)$$

for any terms A and B to transform the query into the simpler, equivalent form:

$$\{x/_{SHIPS} \mid (x.Deadweight > 190) \wedge (x.Shiptype = \text{"supertanker"})\}.$$

This is indeed an interesting result. We started with a constraint on the deadweight of ships, and found that we could add a constraint on their shiptype. If the Shiptype attribute is indexed, the new query may be much less expensive to process. The transformation corresponds to our intuition in this case, as it should. The integrity constraint we used says that all ships with deadweight over 190 thousand tons are supertankers. The end result looks like a simple application of modus ponens, but it is more than this; it is a transformation that depends on properties of the database when viewed as a model of the integrity constraints.

3.5 Formal definition of semantic equivalence transformations

We now develop a general, formal definition of the type of transformation illustrated by the foregoing example. The idea behind the definition is also seen in the example. The transformation should permit us to combine an integrity constraint and a query in such a way that the meaning of the query is not changed and so that terms can be further combined by the application of logical equivalences. Our discussion has three parts: transformation of a well-formed formula (wff) of the typed predicate calculus by means of merging with a second wff; conditions under which the new wff is semantically equivalent to the transformed wff; and the application of this type of transformation to queries and semantic integrity constraints in the relational calculus.

3.5.1 Merging of well-formed formulas

Consider two wffs, X and Y, of a typed, first-order predicate calculus. Suppose that X has the free variables $(x_1, x_2, ..., x_n)$ and that Y has the free variables $(y_1, y_2, ..., y_m)$. Each variable x_i and y_j is *typed*; that is, it ranges over a specified domain. We can write X and Y in terms of predicates P and Q as follows:

$$X = P(x_1, x_2, ..., x_n)$$

$$Y = Q(y_1, y_2, ..., y_m).$$

Under these circumstances, we have the following condition for merging X and Y:

Formula Y can be **merged** *into formula X if and only if the variables $(y_1,...,y_m)$ can be put into one-to-one correspondence with a subset of the variables $(x_1,...,x_n)$ so that corresponding variables range over the same domain. If this condition holds, then formulas X and Y are said to be* **merge-compatible.**

Let x_j' be the variable in X that corresponds to variable y_j in Y (x_j' is not necessarily the same variable as x_j). Then Y can be rewritten as:

$$Y = Q(x_1', x_2', ..., x_m').$$

We now take the *conjunction* Z of X and Y:

$$Z = P(x_1, x_2, ..., x_n) \wedge Q(x_1', x_2', ..., x_m').$$

But the variables $(x_1', x_2', ..., x_m')$ are a (possibly rearranged) sublist of the variables $(x_1, x_2, ..., x_n)$, so we can write Z just in terms of the latter variables:

$$Z = P(x_1, x_2, ..., x_n).$$

We say that the formula Z is the *transformation* of the formula X when merged with the formula Y.

3.5.2 Semantic equivalence of transformed formulas

Let us assume that each variable x_i ranges over some domain of values D_i. Let us further assume that there is some set I of *permitted interpretations* of the the variables $(x_1,... x_n)$, where an interpretation is an assignment of a value from domain D_i to the corresponding variable x_i, for all i from 1 through n. The set I is a subset of the Cartesian product of the domains, denoted by $D \equiv D_1 \times D_2 \times ... \times D_n$. Under these assumptions, we have the following definition:

Two well-formed formulas F1 and F2 over free variables $(x_1,...,x_n)$ are **semantically equivalent with respect to the permitted interpretations** *if and only if F1 and F2 have the same truth value in every permitted interpretation.*

Note in particular that it is not necessary for F1 and F2 to have the same truth value for possible interpretations in D that are not in the subset I of permitted interpretations (as they would have to be if they were *logically* equivalent). We expect that D is reduced to I by means of semantic integrity constraints.

The original wff X and the transformed wff Z of Section 3.5.1 range over the same set of variables. Under what conditions are they semantically equivalent according to the definition just given?

Formula Z is the conjunction of formulas X and Y. It is clear, therefore, that Z is semantically equivalent to X if and only if the formula Y is true under all permitted interpretations of the variables. Now, Y is defined only in terms of the variables $(y_1,...,y_m)$, a subset of the variables $(x_1,...,x_n)$. Hence, every interpretation of $(x_1,...,x_n)$ includes an interpretation of $(y_1,...,y_m)$. Therefore,

The conjunction of two merge-compatible formulas X and Y is semantically equivalent to formula X, if and only if formula Y is true in all permitted interpretations of its variables (y_1,...,y_m). For our purposes, we call this the validity *requirement.*

There is one more important point to consider. Suppose X is actually the quantifier-free matrix of a quantified well-formed formula F_x. Some or all of X's variables will then be bound and not free. Call the bound variables b_i and the free variables f_j. Then F_x can be written as

F_x: $(Q_1b_1)(Q_2b_2) \ldots (Q_hb_h)$ X, or

F_x: $(Q_1b_1)(Q_2b_2) \ldots (Q_hb_h)$ $P(b_1,\ldots,b_h,f_1,\ldots,f_g)$,

where each (Q_ib_i) is either of the two quantifier expressions $\forall b_i$ and $\exists b_i$. It is clear, however, that the quantified well-formed formula F_z formed by substituting Z for X in F_x has the same truth value as F_x for all permitted assignments of values to the free variables f_1 through f_g.

3.5.3 Transformation of a query using a semantic integrity constraint

We now connect the discussion with our central interest in queries and semantic knowledge. Here, the role of the formula to be transformed, F_x, is assumed by a database query. The role of the merging formula Y is played by a semantic integrity constraint. The resulting formula F_z is the new semantically equivalent query.

We draw upon the view of a database in terms of relational calculus, described in Section 3.3. Let $P(b_1,\ldots,b_m,f_1,\ldots,f_n)$ be a well-formed formula of the *tuple relational calculus* [Pirotte78] with free variables b_1 through b_m and f_1 through f_n. Every variable is understood to range over the tuples of a single relation. As before, let (Q_ib_i) be either of the two quantifier expressions $\forall b_i$ and $\exists b_i$. Then any query can be expressed in the form:

Q: $(Q_1b_1)(Q_2b_2) \ldots (Q_mb_m)$ $P_q(b_1,\ldots,b_m,f_1,\ldots,f_n)$,

Considering the query Q as a whole, variables b_1 through b_m are bound, and variables f_1 through f_n are free. There are two kinds of queries to consider. In a *closed* query, there are no free variables ($n = 0$). The answer to a closed query is a yes/no answer, depending upon whether or not Q is true with respect to the current interpretation, that is, the current contents of the extensional database (EDB). If there are free variables ($n > 0$), then the query is an *open* query. The answer to an open query is the set of assignments to the free variables f_1 through f_n that make Q true in the current interpretation. Because variables range over the tuples of relations, the answer to an open query is a set of n-tuples of relation tuples. An open query need not have any quantifier expressions; it must have free variables. In either case, provision must also be made to extract tuple attributes for comparison or retrieval purposes.

As noted in Section 3.3, a semantic integrity constraint can be represented as a closed well-formed formula of the relational calculus. Hence, we can express a constraint as:

$$C: (Q_1c_1)(Q_2c_2) \ldots (Q_kc_k)\, P_c(c_1,\ldots,c_k)$$

where there are no free variables in C taken as a whole. Constraint C has the very important property that it evaluates to "true" in all permitted states of the database; indeed, that is the definition of semantic integrity enforcement.

As we stated above, we want to have query Q and constraint C play the roles of formulas F_x and Y of Section 3.5.1, respectively. It is evident that a query Q is very much like the kind of formula F_x given above. The only additional specification is that the variables range over the tuples of database relations. However, the correspondence between a semantic integrity constraint C and the formula of type Y is not so immediate. We must confront the fact that constraint C has no free variables as it now stands, so it can't be merge compatible with another formula.

We remedy the absence of free variables in C in such a way that we insure the validity requirement stated in Section 3.5.2. Namely, we allow any universal quantifier in C's prefix to be dropped. If the quantifier $\forall c_i$ is dropped, then C can now be expressed as the formula $P'_c(c_i)$, a formula with no prefix and the single free variable c_i. The resulting formula must be true in all permitted interpretations (assignments of a value to variable c_i). This is because the original universally quantified constraint says precisely that the formula is true for all values of variable c_i.

A universal quantifier can be dropped wherever it appears in the prefix, even if it appears within the scope of an existential quantifier. This can be seen from the logical theorem

$$(PREF_1)(\exists x)(\forall y)(PREF_2)P(z_1,x,y,z_2) \rightarrow (PREF_1)(\forall y)(\exists x)(PREF_2)P(z_1,x,y,z_2)$$

where $(PREF_1)$ and $(PREF_2)$ stand for portions of the prefix. This means that a universal quantifier can be "moved left" outside the scope of an enclosing existential quantifier, hence outside the scope of any existential quantifier.

We do not permit an existential quantifier to be dropped. To see why we impose this restriction, consider what it would mean to do so. The variable bound by the quantifier would now be free. What tuples in the range of the variable would satisfy the resulting formula? We have no way to tell. All we know is that at least one tuple does satisfy the formula, but we cannot assert that the formula is true for every such assignment.

It must be noted of course that the requirement of merge compatibility means that we can only create free variables in the constraint for which there is a corresponding free variable in the matrix of the query.

We now have a direct parallel to the process set forth in Sections 3.5.1 and 3.5.2. We summarize the process for transforming a query into a semantically equivalent query as follows:

Let Q be a query expressed in the tuple relational calculus:

$$Q: (Q_1b_1)(Q_2b_2) \ldots (Q_mb_m)\, P_q(b_1,\ldots,b_m,f_1,\ldots,f_n),$$

where every variable is understood to range over the tuples of a single relation and (Q_ib_i) is either of the two quantifier expressions $\forall b_i$ and $\exists b_i$.

Let C be a semantic integrity constraint represented as a closed well-formed formula of the tuple relational calculus:

$$C: (Q_1c_1)(Q_2c_2) \ldots (Q_kc_k)\, P_c(c_1,\ldots,c_k)$$

where C has the property that it evaluates to true in all permitted states of the database. Let (c_a, c_b, ..., c_i) be a subset of the universally quantified variables of constraint C, and let $P'_c(c_a, c_b, \ldots, c_i)$ be the well-formed formula produced by dropping the corresponding universal quantifiers. Then, if and only if the variables (c_a, c_b, ..., c_i) can be put into one-to-one correspondence with a subset of the variables ($b_1,\ldots,b_m,f_1,\ldots,f_n$) so that corresponding variables range over the same relation, it follows that the query Q'given by:

$$Q': (Q_1b_1)(Q_2b_2)\ldots(Q_mb_m)\, P_q(b_1,\ldots,b_m,f_1,\ldots,f_n) \wedge P'_c(c_a, c_b,\ldots,c_i)$$

is semantically equivalent to query Q; that is, Q'gives the same answer as Q in every permitted state of the database. For convenience, the newly transformed query can be written as:

$$Q': (Q_1b_1)(Q_2b_2) \ldots (Q_mb_m)\, P'_q(b_1,\ldots,b_m,f_1,\ldots,f_n)$$

where P'_q is the conjunction of P_q and P'_c.

3.6 Logical transformations in semantic query optimization

A semantically equivalent query formed according to the preceding definitions may well be more expensive to process than the original query. After all, the new query apparently involves more terms than the original. However, various improvements in efficiency may arise by a further transformation or simplification of the new expression, based upon the replacement of terms by terms that are logically equivalent. The effect is that terms in the new qualification expression are subject to cancellation or combination. Simplifications can be based upon such domain-independent properties as transitivity of numerical comparators, along the lines suggested by Youseffi [Youseffi78].

Of great importance are simplifications that involve semantic integrity constraints in the form of implications. To see this, consider a constraint of the form

$$\forall x\, P(x) \rightarrow Q(x)$$

where the variable x ranges over some relation R. Suppose the matrix of some query Q contains the term P(z) where the variable z ranges over the same relation as the variable x in the constraint. According to the procedures outlined in the preceding sections, we can transform Q into a semantically equivalent query Q'whose matrix contains the conjunction:

$$P(z) \wedge (P(z) \rightarrow Q(z)).$$

However, by the logical equivalence

$$(\Lambda \wedge (\Lambda \rightarrow B)) \equiv (\Lambda \wedge B)$$

we can replace this conjunction by the simpler conjunction

$P(z) \wedge Q(z)$.

The net effect is as if the original query condition P(z) were used to infer the new condition Q(z) by means of the semantic integrity constraint. Similarly, if the original query contains the term $\neg Q(y)$ where the variable y ranges over relation R, then by the equivalence

$$(\neg B \wedge (A \rightarrow B)) \equiv (\neg B \wedge \neg A)$$

this term can be replaced by the conjunction $\neg Q(y) \wedge \neg P(y)$. Indeed, if the original query actually contains the conjunction $P(z) \wedge Q(z)$, z ranging over relation R, then by using the logical equivalence

$$(A \wedge B \wedge (A \rightarrow B)) \equiv A$$

we can replace this conjunction by the simple condition P(z). In other words, the condition Q(z) has been shown to be derivable from P(z), hence it is superfluous and may be dropped from the query.

This concludes our general discussion of the formal basis for semantic query optimization. In the next chapter, we describe the QUIST system, in which these ideas have been implemented and tested.

Chapter 4
The QUIST system

In Chapter 3, we presented the formal basis for the transformation of one relational database query into another semantically equivalent query. This semantic equivalence transformation is at the heart of semantic query optimization. In this chapter, we take up the issue of creating an effective semantic query optimization system, and we describe the operation of QUIST, an implemented semantic query optimization system.

In Section 4.1, we discuss the factors that influence the effectiveness of a semantic query optimization system, particularly the choice of what semantic knowledge should ever be considered for semantic query optimization, and the way that structural and processing knowledge is used to control the semantic transformation of queries. We begin the description of QUIST in Section 4.2, noting the class of queries it handles and the types of semantic rules it uses. In Section 4.3 we present an overview of system operation. We indicate that QUIST operates in a plan-generate-test mode in which the problem of query optimization is addressed at different levels of abstraction. Finally, in Section 4.4 we discuss the actions of the system in great detail by means of an example. We show how the generalizations about processing queries to relational databases discussed in Chapter 2 are incorporated in specific heuristics. We show specifically how the heuristics are used to control which knowledge base rules are used for query transformations, and we relate the heuristics in general to particular types of transformations of relational database queries.

4.1 The design of an effective semantic query optimization system

A query optimization strategy based upon semantic equivalence transformations presents both opportunities and dangers. The opportunities lie in the possibility of eliminating unneeded operations, or replacing or modifying operations with more efficient ones. The dangers arise from what may be a large store of semantic integrity constraints. Any query might possibly be transformed by any combination of those constraints. If not controlled in some way, the process of generating transformations of the query can be very expensive.

There are two ways to bring the process under control: by restricting what semantic integrity constraints will ever be considered for query transformation, and by using knowledge of database structure and processing to guide the transformation of any particular query.

4.1.1 Choosing semantic knowledge

The kinds of knowledge that are most useful for semantic query optimization depend primarily on two factors: the kinds of queries that are to be handled, and the physical organization of the data.

The most common kinds of queries involve access between entities and their attributes. A typical query may be "What is the length of the Totor?" in which an entity's name (or *identifier*) is given and some attributes are sought. The direction of access is reversed in a query like "What are the names of the French ships over 300 feet long?" in which constraints are specified on the values of attributes and the retrieval of entity identifiers is sought. Both directions of access are combined in a query such as "List the draft of French ships" where a set of entities is specified by means of constraints on one set of attributes, and retrieval of another set of attributes is sought. When the query involves relationships and not just objects, access between entities and attributes is still crucial. For instance, in the query "Which Italian ships are commanded by admirals?" the set of ship captains who are considered as commanders is confined to those whose rank (an attribute) meets a specified constraint.

The importance and frequency of these queries is reflected in the physical organization of databases, the second major factor that influences what semantics should enter into semantic query optimization. A prime objective of semantic query optimization is to produce useful constraints. As pointed out in Chapter 2, the physical structure of a relational database is typically organized into records and fields that correspond to entities and attributes. In anticipation of queries with constraints on attributes, indexes are stored that contain pointers to physical locations of records (entities) with particular values in certain fields (attributes). Constraints on indexed attributes obviously are useful, as are constraints on attributes of entities that have links to other entities.

In consideration of both the common kinds of queries and the typical physical organization of databases, it is evident that constraints on the attributes of entities are of utmost importance. The kind of semantic rules that are most useful are rules that relate constraints on attributes expressed in queries with constraints that are useful in the sense just described.

This observation leads to the view of semantic query optimization as a movement of constraints among different parts of the database. One kind of semantic rule that directly supports the movement of constraints is what Kent [Kent78] calls general *restrictions on relationships*. These are constraints on the participants in relationships that are more specific than simply designating their entity type. They relate properties or attributes of one participant with properties or attributes of another. One such kind states

$$C1\ \theta\ C2$$

where C1 and C2 are simple restrictions on attributes and θ is a Boolean comparator such as less-than or greater-than. For example, there may be a relationship between a consignment of cargo and the insurance policy that covers it, to the effect that the amount of the policy does not exceed the value of the consignment. In this case, there is a relationship between the amount attribute of the policy and the value attribute of the consignment. Another kind of rule that restricts relationships states

$C1 \rightarrow C2$

for constraints C1 and C2. For instance, we may know that only leasing companies own ships with a deadweight (size) over some amount. That is, given a certain constraint on a ship's deadweight, another constraint can be inferred on the type of business of the company that owns the ship.

4.1.2 Controlling transformations with structural and processing knowledge

There is no guarantee that any semantic equivalence transformation leads to a lower cost query. Indeed, a competent database administrator chooses database file structures that support efficient access to frequently referenced data. Thus, assuming that a conventional query optimizer is used, it is reasonable to expect that many queries can be answered efficiently in the form in which they are posed.

Therefore, an effective semantic query optimization system must determine whether to seek cost reductions via semantic transformations. If it does, it must confine its efforts as much as possible to the transformations that are most likely to result in lower cost queries. It should not undertake costly efforts only to find that a reasonably efficient query cannot be improved further.

As we discussed in Chapter 3, the ability to carry out semantic equivalence transformations rests on the semantic knowledge about the database. As we shall see in this chapter, the ability to control the semantic query optimization system depends upon knowledge about what transformations are likely to yield a lower cost query. This ability rests in turn upon two kinds of knowledge: knowledge of the physical file organization of the database, and knowledge of the available retrieval processes, particularly in terms of how various aspects of those processes influence their cost.

In Chapter 2 we indicated that one of the main results of conventional query optimization is the identification of standard file structures and an appreciation of the factors that contribute to the cost of query processing. We can see how this interacts with judging the potential usefulness of a semantic transformation. Consider the query "What ships are carrying iron ore?" posed to a database that lists information about ships and their current cargoes. Three kinds of information can be brought to bear to decide the usefulness of semantic transformations in this case.

- Knowledge of processing cost factors. Two ways to extract qualifying tuples from a relation are to perform a sequential segment scan and to perform a scan by way of a clustered index. The latter method is usually much less expensive. This means that the presence of a restriction on a clustered index attribute significantly lowers the cost of this kind of process.

- Knowledge of file structures. In this case, let us assume that there is a SHIPS relation stored as one file, and that the file has a clustered index on the Shiptype attribute.

- Knowledge about the semantics of the database. Let us assume that there is a semantic integrity constraint to the effect that the only type of ship capable of carrying iron ore is a bulk ore carrier. That is, no tuple can exist in the SHIPS relation for which the Cargo

field has the value "iron ore" and the Shiptype field has some value other than "bulk ore carrier".

From the knowledge of processing cost factors, an effective semantic query optimization system should rate as potentially useful any transformation that starts with a query that must be processed by a segment scan and that results in a query that can be processed by an indexed scan. From the knowledge of file structures, the system should determine for this query that there is a potential opportunity to make this kind of transformation. What is needed is a semantic constraint that relates the values of the Shiptype and Cargo attributes. The knowledge of the semantics of the database gives such a constraint in this case. The query can be transformed into "What bulk ore carriers are carrying iron ore?" The cost to process this query should be compared to the cost of the original query to select the one to be posed to the database.

This example suggests how the flow of information and control can be organized in an effective semantic query optimization system. The system analyzes the query with respect to processing methods and file structures. The analysis identifies potentially useful transformations specialized to the context of the current query. That is, they are expressed in terms of relations or attributes that are involved in the query. If potentially useful transformations are identified, the system retrieves appropriate semantic constraints using the specialized descriptions. The system then carries out semantic equivalence transformations and simplifications with those constraints. Finally, the system evaluates the efficiency of the resulting queries and selects for processing the one with lowest estimated cost.

4.2 Introduction to the QUIST system

The QUIST system (QUery Improvement through Semantic Transformation) is a program that has been implemented to explore the design and operation of an effective semantic query optimization system in the context of an important class of relational database queries. The system demonstrates the ability to transform queries by reasoning about the semantics of the database. It shows that it is possible for a semantic query optimization system to achieve significant improvements in query processing efficiency that are unattainable by conventional methods. It also shows that a semantic query optimization system can run with acceptable overhead compared to the overall cost of processing queries. In doing so, QUIST demonstrates the use of specific inference guiding heuristics based on structure and processing expertise originating in conventional query optimization research.

In this section, we describe the class of queries for which QUIST can attempt semantic query optimization. We also indicate the kinds of semantic integrity rules that QUIST can use for this purpose. The choice of the kinds of semantic knowledge used by QUIST follows the ideas set forth in Section 4.1. We take up the other issue of Section 4.1, the control of semantic transformations by means of structural and processing knowledge, later in this chapter.

4.2.1 The class of queries handled by QUIST

The QUIST query language is a query language for relational databases. It is in a class of languages that can be termed *attribute/constraint* languages. This choice reflects the importance of constraints on attributes as described in Section 4.1. Indeed, the entire QUIST system is designed from the point of view that the most useful semantic transformations in relational queries can be seen as the addition, deletion, or modification of constraints on database attributes.

Attribute/constraint languages are particularly simple, hence somewhat limited, yet have been shown to admit a significant subset of restrict-join-project relational queries (see Section 2.3). Two examples of attribute/constraint query languages are the IDA language developed by Sagalowicz at SRI International [Sagalowicz77] and the APPLE language developed by Carlson and Kaplan at Northwestern University [Carlson76]. The QUIST query language is modelled most closely on IDA. In the context of the LADDER natural language database access system [Hendrix78], IDA has been shown to admit a substantial and interesting class of queries.

The essential distinguishing feature of an attribute/constraint language is that it presents a relational database as if it contained just a single virtual relation, masking the real relations underlying it. The point of this is to make the specification of relational database queries as simple as possible. It buffers users and natural-language understanding programs from the need to know the structure of the database and from any reorganization of the database that involves changes in the association of attributes with relations.

The single virtual relation is formed from the real relations as follows. A subset of all the possible joins between relations is specified such that at most one join is permitted between any two real relations, and so that there exists one and only one logical path (sequence of joins) between any two real relations. The set of joins is performed and duplicates are eliminated. The result is the virtual relation. If the virtual relation is thought of as a graph whose nodes are the real relations and whose edges are joins between real relations, then the virtual relation is a tree structure of real relations. Any query to the database involves a subtree of this virtual relation.

The virtual relation makes possible a great simplification in the specification of restrict-join-project queries: joins are made implicit because they have already been specified in the definition of the virtual relation That means that an attribute/constraint query is specified solely in terms of restrictions and projections. In other words, an attribute/constraint query consists of boolean combinations of simple constraints on attributes, plus a list of attributes whose values are desired. Significantly, tuple variables need no longer be used in the query, because there is only one (virtual) relation.

The cost of this simplification is a set of limitations on the general relational model. For one thing, attribute names must be unique throughout all relations because tuple variables are no longer available to distinguish them. For another, no join is permitted other than those prespecified through the definition of the virtual relation. The latter limitation implies that a relation can only be involved

once in a query (for instance, it cannot be joined to itself). With respect to the concepts represented in the database, this means that it is possible to represent only one kind of relationship between any two classes of entities represented as relations. Moore discusses some of these limitations in [Moore79]. Nevertheless, as indicated above, attribute/constraint languages permit the expression of an important and useful range of queries.

The level of abstraction presented by an attribute/constraint query language is illustrated by an example from IDA. Suppose a database contains two relations:

SHIP: (Shipname Shipclass Shiptype)

SHIPCLASS: (Class Type Length Draft)

where, for instance, Shipname is the name of a ship and Length is the length of any ship in a particular ship class. Assume the choice is to permit SHIP and SHIPCLASS to be joined on Shipclass and Class, respectively.

A request for the names of all ships is merely a request to print all values of the Shipname attribute. No constraints need to be specified. In IDA, this request is specified as (? Shipname). In general, an expression of the form (? Attribute) returns the value of the specified attribute. To request the length of a ship whose name is "Totor", it is necessary to place a constraint on the Shipname attribute and to request the value of the Length attribute. The IDA specification is:

(Shipname = "Totor") (? Length).

The two attributes are on separate underlying relations, but IDA hides this from the user. The IDA query processing system determines the logical access path between the two relations. It looks up the prespecified join between SHIP and SHIPCLASS and, in effect, transforms the joinless form of the query into one that includes the join term (SHIP.Shipclass = SHIPCLASS.Class).

The class of queries handled by QUIST is actually somewhat different from IDA's. The qualification of a QUIST query is a conjunction of constraints on attributes, rather than a general boolean combination of constraints. This limitation is compensated for by permitting the constraint on each attribute to be, in effect, a disjunction of simple constraints. QUIST does not attempt to perform semantic transformations on such questions as "What ships are registered in France or are over 200 feet long?" where the disjunction involves constraints on more than one attribute. In this case, the design decision was to avoid the added difficulties of inference with general disjunctions on the grounds that many practical queries do not involve them and because of the low probability of finding less expensive transformations of them.

As with IDA, QUIST queries can specify constraints on numerical-valued attributes and constraints on string-valued attributes. A numerical constraint is specified as the intervals in which the attribute's value is permitted to fall. The complete constraint can be a disjunction of these intervals. For instance, suppose a query constrains the Age attribute in a personnel database to be greater than 20 and less than or equal to 25, or to be greater than or equal to 65 and less than 70. This constraint is specified as

(Age ∈ ((20 25] [65 70))).

This constraint is considered to be a disjunction of two intervals. QUIST checks that intervals do not conflict. If the constraint on a numerical attribute is in fact a simple constraint, such as specifying that Age is less than 65, then the preceding form can be abbreviated as

(Age < 65)

rather than, for instance, specifying an interval one of whose bounds is $+\infty$ or $-\infty$.

String-valued attributes can be constrained to be a member of some set of strings, or to be excluded from some set of strings. For example, if Shiptype must be either "tanker" or "fishing", the constraint is specified as:

(Shiptype ∈ {"tanker" "fishing"}).

Another type of constraints for string-valued attributes is typified by the constraint that Shiptype must be neither "bulk" nor "refrigerated":

(Shiptype ∉ {"bulk" "refrigerated"}).

This is equivalent to a conjunction of simple inequality constraints. As with numerical constraints, the notation for a simple constraint can be abbreviated, as for example:

(Shiptype = "supertanker")

to indicate that the Shiptype must be a supertanker.

The complete syntax of queries admitted by the QUIST system is given in Appendix A.

4.2.2 QUIST's semantic knowledge base

The QUIST system captures the important semantic integrity restrictions on attributes and relationships described in Section 4.1. The single-relation view of the database makes it easy to express these restrictions, subject to the limitation that only one kind of relationship can be represented between any two kinds of entities. The restrictions are stored in a "conceptual schema" or knowledge base where they are associated with the attributes they mention.

The simplest type of restriction is what McLeod [McLeod76] refers to as *domain definition.* This type of restriction specifies the possible values of an attribute regardless of the values of any other attributes, and regardless of any relationships involving the entity to which the attribute is associated. For instance, if it is known that all ships in the database have a deadweight of between 20 thousand tons and 450 thousand tons, regardless of their shipclass, their registry, the type of business of their owner, or any other factor, then the knowledge base would associate with the Deadweight attribute the restriction:

(Deadweight ∈ ([20 450])).

In terms of the more general first-order formulas described in Chapter 3, domain definition restrictions are implicitly universally quantified over the (real) relation to which the restricted attribute is associated. Thus, the example restriction corresponds to:

$$\forall x/_{SHIPS}\ (x.Deadweight \geq 20) \wedge (x.Deadweight \leq 450)$$

The other types of restrictions involve two or more attributes. Two kinds of multiattribute restrictions are represented. One kind, called a *bounding rule*, asserts that the value of one attribute is bounded by the value of another attribute. For example, the quantity of a cargo that can be carried by a ship is bounded by the capacity of the ship. If there are two relations, CARGOES and SHIPS, and the unique logical access path defined between them corresponds to a "carrying" relationship, then the bounding rule can be represented simply as:

(Quantity ≤ Capacity).

The corresponding form of this restriction in terms of a general first order formula is a universally quantified expression in which the predefined logical access path between SHIPS and CARGOES is made explicit.

$$\forall x/_{SHIPS}\ \forall y/_{CARGOES}\ (x.Shipname = y.Ship) \rightarrow (y.Quantity \leq x.Capacity)$$

The other type of multiattribute semantic restriction is called a *production*. A production is a rule of the form:

$$C_1(A_1) \wedge C_2(A_2) \wedge \ldots \wedge C_k(A_k) \rightarrow C'(A).$$

Every term in the rule is a constraint expression on an attribute. No attribute can appear more than once on the left hand side. An example of a production is a rule that states that cargoes of refined petroleum products are carried only by ships whose deadweight is under 60 thousand tons. This rule involves the same "carrying" relationship and hence the same implicit join between CARGOES and SHIPS as in the previous example. In this case, the rule is represented as:

(Cargotype = "refined") → (Deadweight ≤ 60)

where Deadweight is in units of thousands of tons. The production form used by QUIST is the *Horn clause* form common in deductive databases [Nicolas78a].

As with bounding rules, the corresponding general first order formula is a universally quantified expression with an explicit join term when attributes from more than one relation are involved:

$$\forall x/_{SHIPS}\ \forall y/_{CARGOES}\ (x.Shipname = y.Ship) \wedge (y.Cargotype = \text{"refined"}) \rightarrow (x.Deadweight \leq 60)$$

The fact that the semantics of domain definitions, bounding rules, and productions can be expressed as simply as in the foregoing examples is one of the motivations behind the choice of the QUIST data model and language.

4.3 Overview of the operation of the QUIST system

The QUIST system accepts a query in the QUIST relational database query language, produces a set of semantically equivalent queries (possibly including only the original query), and returns the query from that set with the lowest estimated retrieval cost. In this section, we present an overview of how QUIST performs these tasks. We defer detailed descriptions until Section 4.4.

In Section 1.1, we indicated that QUIST's operates in a mode of *plan, generate, and test* that appears in other artificial intelligence programs for solving a wide range of problems [Feigenbaum71]. The purpose of the planning step is to identify both desirable and undesirable characteristics of a solution to the given problem. These characteristics of a solution are used to control the generation step in which candidate solutions are produced. Finally, the testing step carries out detailed evaluation of the candidate solutions in order to select the one with highest merit. Overall, the three steps are characterized by the degree of abstraction at which the problem is addressed, and by the kind of search carried out.

4.3.1 The planning step -- identification of constraint targets

The planning step starts with the constraints specified in the input query. Using heuristics based on structure and processing knowledge, the system determines which database relations, if any, are *constraint targets*. A relation that is a constraint target is one that has attributes on which additional constraints should be sought. Constraint targets are determined by viewing the query only in terms of which relations it involves, either through constraints or through selection for output. The search space is very simple, consisting merely of assignments of relations to the sets of targets and nontargets. Incidentally, the concept of constraint targets should not be confused with the term *target list*, commonly used to describe which attributes are to be output from the database. Instead of target list, we use the term *output attributes*.

If there are no constraint targets, QUIST merely returns the original query unchanged. In such a case, QUIST has determined that it is not worthwhile to generate equivalent queries because no equivalent query is likely to cost less to process than the original query. On the other hand, if there are constraint targets then QUIST continues on to the generation of semantically equivalent queries.

4.3.2 The generation step -- production of constraints and semantically equivalent queries

The generation step consists of a cycle of constraint production operations repeated until no more constraints are produced. Each cycle of constraint production retrieves relevant knowledge base rules, filters them according to structurally-based criteria (that is, the list of constraint targets), tests them for applicability against the current constraints, and asserts new constraints if possible. The process terminates when some cycle fails to generate new constraints.

The generation step treats the query less abstractly than the planning step does. It must use the precise constraints on database attributes, not merely the names of constrained or output attributes. The search at the generation level is through a space of semantically equivalent queries. Each move consists of the production of another constraint. Only plausible moves are permitted because the constraint target list produced by the planning step permits only those those transformations that may possibly lower the cost of processing.

4.3.3 The testing step -- selection of the query with lowest estimated cost

The generation step produces one or more QUIST queries that are known to produce the same answer. In the testing step, each query is analyzed by conventional query optimization methods. This yields an estimated lowest cost to perform each query. The query with the minimum estimated lowest cost is determined.

At the testing level, the search is through a space of physical realizations of a single logically expressed query. The query itself must be analyzed in the greatest detail, in terms of the actual database files it accesses and the sequence in which it accesses them.†

4.3.4 Summary of QUIST operations

In describing the detailed operation of the QUIST system, it is convenient to distinguish the three steps just described plus the task of grouping inferred constraints into semantically equivalent queries. To summarize the operations, then, the following steps take place:

1. Identification of constraint targets (the planning step)
2. Inference of new constraints (part of the generation step)
3. Grouping of constraints into the set of semantically equivalent queries (conclusion of the generation step)
4. Estimation of the minimum processing time for each query and selection of the query with the lowest estimated processing time (testing step)

We now present an example to illustrate these steps.

---

†The problem is nonetheless abstracted in the sense that the actual query is not carried out; rather, the cost to perform it is just estimated.

4.4 Example of the operation of the QUIST system.

In this section, we begin an example of the operation of the QUIST system. The example brings out the kinds of knowledge that semantic query optimization requires, and shows precisely how QUIST integrates the different knowledge sources into an effective system. In particular, the example is used to identify a set of heuristics that guide the inference of new constraints. These heuristics are specific to relational databases as described in Chapter 2.

The example is specifically tailored to illustrate the special capabilities of semantic query optimization. In particular, it involves both the addition and the elimination of relations from a query. The example also illustrates how structural knowledge is used to halt a particular line of constraint generation when the constraints appear to offer no hope of reducing the cost of query processing.

QUIST's operation is illustrated using the relational database illustrated in Figure 4-1:

SHIPS (Shipname Owner Shiptype Draft Deadweight Capacity Registry)

PORTS (Portname Country Depth Facilitytype)

CARGOES (Ship Destination Shipper Cargotype Quantity Dollarvalue Insurance)

OWNERS (Ownername Location Assets Business)

POLICIES (Policy Issuer Coverage)

INSURERS (Insurer Insurercountry Capitalization)

Figure 4-1: Example database relations

QUIST operates with an attribute/constraint data model. Specifically, it treats the database as a single virtual relation. It is therefore necessary to specify the unique logical access paths among the real relations. The joins that underlie the permitted logical access paths are:

1. OWNERS.Ownername = SHIPS.Owner

2. SHIPS.Shipname = CARGOES.Ship

3. CARGOES.Destination = PORTS.Portname

4. CARGOES.Insurance = POLICIES.Policy

5. POLICIES.Issuer = INSURERS.Insurer

which stand for, respectively,

1. A ship and its owner

2. A ship and a cargo it is carrying

3. A cargo and its destination port

4. A cargo and the policy that insures it

5. A policy and its issuing company

The database is assumed to be implemented by means of one file per relation. It is further assumed that the SHIPS file has a clustering index on its OWNER attribute. This means that the SHIPS file is clustered with respect to the OWNERS file; given a tuple in the OWNERS file, the corresponding tuples in the SHIPS file (that is, the ships owned by that owner) can be accessed much less expensively than by a sequential search of SHIPS. In addition, the OWNERS file is much smaller than the SHIPS file.

A very simple knowledge base of general semantic rules accompanies the database in this example. We don't wish to claim the validity of all these rules; they are merely useful illustrations of the kinds of rules that can be used for semantic query optimization. The rules are:

- Rule R1. Every ship over 350 thousand tons deadweight can operate only at ports with offshore load/discharge capabilities.
- Rule R2. Only leasing companies own vessels that exceed 300 thousand tons deadweight.
- Rule R3. A cargo is never insured for more than its dollar value.
- Rule R4. A ship carries no more cargo than its rated capacity.
- Rule R5. Any cargo other than liquefied natural gas or refined petroleum products that is worth more than 500,000 dollars is handled only at general cargo ports.
- Rule R6. The only ships whose deadweight exceeds 150 thousand tons are supertankers or aircraft carriers.
- Rule R7. Cargoes worth over three million dollars and carried by supertankers are insured by policies issued by Lloyds.
- Rule R8. Ships owned by petroleum companies only carry liquefied natural gas, refined petroleum products, or crude oil.

The rules in the example knowledge base are represented to QUIST as:

R1: (Deadweight > 350) → (Facilitytype = "offshore")

R2: (Deadweight > 300) → (Business = "leasing")

R3: (Coverage ≤ Dollarvalue)

R4: (Quantity ≤ Capacity)

R5: (Cargotype ∉ {"LNG" "refined"}) ∧ (Dollarvalue > 500) → (Facilitytype = "general")

R6: (Deadweight > 150) → (Shiptype ∈ {"supertanker" "carrier"})

R7: (Dollarvalue > 3000) ∧ (Shiptype = "supertanker") → (Issuer = "Lloyds")

R8: (Business = "petroleum") → (Cargotype ∈ {"LNG" "refined" "oil"})

The subject of our example is the following query:

> ***"List the destination of cargoes worth less than one million dollars being carried by supertankers over 400 thousand tons deadweight to ports with offshore load/discharge facilities."***

Note that this query was invented specifically to illustrate semantic query optimization capabilities. As motivation, consider a shipping analyst who wishes to detect cases in which very large ships are being employed wastefully so that they can be rerouted to more profitable activities.

The representation of this query to *QUIST* is:

Q: (Deadweight > 400) ∧ (Shiptype = "supertanker")
∧ (Dollarvalue < 1000) ∧ (Facilitytype = "offshore");
(? Destination)

Processing query Q as given involves three relations, SHIPS, CARGOES, and PORTS, and two joins among them: SHIPS to CARGOES, and CARGOES to PORTS. However, we readily see that semantic rule R1 makes the constraint on Facilitytype superfluous. If this constraint is eliminated, then it won't be necessary to involve PORTS in the processing at all. PORTS is involved in Q only to restrict tuples in CARGOES, and it turns out that this restriction is unnecessary.

Moreover, the constraint on Deadweight also makes it possible to infer a constraint on the Business attribute of the OWNERS file. Although this introduces a join to a new file, the database is structured so that this may be advantageous. This is because this join has, in effect, been precomputed and stored as the link from OWNERS to SHIPS.

In addition, a constraint can be inferred on the Coverage attribute of the POLICIES relation by means of rule R4. However, it is not desirable to involve POLICIES in the query because the join to CARGOES is not supported by a prestored link or index.

We now discuss how QUIST handles this query.

4.4.1 Step 1 - Identification of constraint targets

QUIST's first step establishes inference goals. The task of this goal-setting step is to accept the list of attributes that are constrained or designated for output by the query, and to return a (possibly empty) list of target relations on which the placement of additional constraints may be worthwhile.

In this step, QUIST determines whether it seems worthwhile to seek to transform the given query into an equivalent one. If it does seem worthwhile, then QUIST seeks to identify what opportunities exist for cost-reducing transformations. However, it may be the case that it is not worthwhile to transform the given query. For example, the query restrictions might consist of just a single constraint which happens to be on an attribute with a clustering index. No additional constraints can reduce the processing effort for that query. Any effort devoted to inference would then be wasted. Even if inference is not ruled out, there will probably be only a few relations on which the placement of additional constraints will reduce query processing effort. Pruning the set of target relations can significantly reduce useless inference effort.

To produce a set of constraint targets from a set of constrained or output attributes, QUIST uses *constraint generation heuristics*. These heuristics are based upon knowledge about the structure of the database and about the factors that contribute to the cost of retrieval. The heuristics reflect the expert knowledge developed from analysis of relational database query processing.

By what criterion should target relations be chosen? The general answer is that a relation should be the target for the generation of constraints if and only if the placement of such constraints on the relation makes some retrieval operation less expensive or renders it unnecessary altogether.

We can make this criterion more specific in the context of the retrieval operations for restrict-join-project queries discussed in Section 2.3. The major operations are scanning a relation, and joining two relations.

4.4.1.1 Scanning a relation

We first consider scanning a relation. A relation can be scanned in three ways: by a segment scan, by a scan using a nonclustered index, or by a scan using a clustered index. A segment scan looks at every page in the segment that contains the relation. A scan with a nonclustered index looks (more or less) at one page for every qualifying tuple.

As for a clustered index scan, we introduce the concept of *restriction selectivity* [Yao79]. Selectivity is a fraction between 0 and 1. It corresponds to the fraction of tuples in a relation that meets some constraint. The stronger the constraint, the closer selectivity is to 0. Let attribute A have a clustered index. If constraint C is imposed on A, and C has a selectivity value of RSEL, then a clustered index scan via attribute A using constraint C retrieves approximately a fraction RSEL of the pages on which the relation is stored.

Consideration of these alternatives leads to the generalizations noted in Section 2.5:

- **G1.** A restriction on an attribute that is not indexed leads to an expensive scan.
- **G2.** A restriction (other than an equality predicate) on an indexed attribute where the index is not a physically clustering index leads to an expensive scan.
- **G3.** A restriction on a physically clustering index can be processed efficiently.

These generalizations give us the following inference guiding heuristic:

> **H1. Try to exploit a clustered index.** *Try to obtain a constraint on an attribute of a relation which is restricted in the query and which has a clustered indexed attribute that is not restricted in the query.*

Another heuristic arises from the same generalizations. It involves a *clustering link* between two relations that effectively precomputes and stores the join between them. There is a clustering link from relation X to relation Y if each tuple in relation X has a pointer to the corresponding tuples of relation Y and those corresponding Y tuples are physically grouped together. The actual join can be performed with X as the outer relation and Y as the inner relation. That is, X is scanned and for each qualifying tuple, the pointer gives the corresponding tuples of Y quite inexpensively. The same effect is achieved if Y has a clustering index on the attribute by which it is joined to X.

From the perspective of scanning relation Y, however, the prestored join with X opens another opportunity to reduce retrieval cost. If X is much smaller than Y and if an effective constraint on X can be found, then the clustering link can be followed to extract qualifiers from Y inexpensively. One way to look at this is to regard X as the parent of Y in a hierarchy. Constraining the parent relation is very effective for constraining the child relation.

The surprising aspect is that it can be advantageous to scan Y via a join from X *even if X does not appear in the original query.* That is, the cost of the overall query can be reduced by introducing an additional file and an additional join. This is one case that is clearly contrary to the intuition expressed in conventional query optimization research. The exploitation of a clustering link is expressed in the following heuristic:

> **H2. Push a constraint up a hierarchy.** *A relation should be a constraint target if it has a clustering link into a much larger file that is constrained in the query, even if the relation itself is not in the original query.*

For the most part, however, it is not a good idea to introduce an additional relation and extra join operations into a query for the obvious reason that joins are normally expensive. This advice is summed up in the heuristic:

> **H3. Don't introduce unlinked joins.** *With the exception of the clustering link*

(parent/child) case, do not generate constraints for relations that are not part of the original query.

4.4.1.2 Joining two relations

We now consider the join operation. Regardless of the method chosen to perform the join, we have noted in Section 2.5 that

- G4. The cost of joins generally dominates the overall cost of processing.
- G5. A join between two large and weakly restricted relations is very expensive.

Thus, much of our concern in finding new constraints centers on reducing the cost of joins. We have considered performing joins by two methods: the nested loops method and the merging scans method. For simplicity in QUIST, we assume that all joins are carried out by the nested loops method, but much of the justification for the ensuing inference heuristics holds for either method.

In the nested loop method, one relation (called the *outer* relation) is scanned, and for each outer tuple that meets the constraints on that relation, the second relation (called the *inner* relation) is scanned. The inner scan seeks qualifying inner relation tuples that match the outer tuple on the join attributes. We noted in Chapter 2 that "the cost of the nested loops method is the cost of scanning the first relation plus the product of the number of qualifying first relation tuples with the cost of scanning the second relation." This presents three opportunities to reduce the cost of the join by the generation of constraints: reduce the cost of the outer scan, reduce the number of qualifying outer tuples, and reduce the cost of the inner scan.

We've already discussed how to reduce the cost of scanning a relation, so we take up the question of how the generation of constraints can help to reduce the number of qualifying tuples in the outer scan. Let's first consider the underlying intuition. Suppose two relations X and Y are to be joined and that both are restricted on some of their attributes. From the point of view of X, the join to the restricted relation Y can simply be seen as a somewhat more indirect restriction than the simple constraints on X's attributes. That is, for some tuples in X that otherwise meet the restrictions on X's attributes, there are no corresponding tuples in Y, hence those tuples of X do not participate in the join.

We would like to translate this indirect restriction into a simpler one in terms of constraints on attributes of X so that it can be applied prior to the cross referencing scan that makes the join expensive to perform. A constraint on an attribute can be applied much less expensively than a constraint imposed indirectly through a join.

Let's make this clearer with an example. Suppose we request the owners of French ships carrying cargoes of refined petroleum products:

Q: (Registry = "France") ∧ (Cargotype = "refined"); (? Owner)

This involves a join between SHIPS and CARGOES. The requirement that each SHIPS tuple be joined to a restricted CARGOES tuple can be viewed as another restriction on SHIPS. However, the subset of French ships that are carrying refined products can't be determined prior to performing the join as the query is stated. If SHIPS is the outer relation, there will have to be a scan of the inner relation CARGOES for every French ship.

Now, suppose there is a general rule that only ships with a deadweight under 60 thousand tons carry refined products. This is represented as the QUIST rule:

R: (Cargotype = "refined") → (Deadweight < 60)

The attribute Deadweight is on the SHIPS relation and the attribute Cargotype is on the CARGOES relation. Rule R makes it possible to infer the constraint (Deadweight < 60) "across the join boundary" from the CARGOES relation to the SHIPS relation. The transformed query Q'so obtained is:

Q': (Cargotype = "refined") ∧ (Registry = "France") ∧ (Deadweight < 60); (? Owner)

Instead of having to scan CARGOES for every French ship, it is now only necessary to scan CARGOES for every French ship of less than 60 thousand tons deadweight. This should bring about a substantial reduction in the cost of performing the join.

Reduction in the number of qualifying tuples is limited to the movement of constraints across the join boundary. No reduction is achieved if the inferred constraint depends entirely on constraints on the same relation. This is because every tuple in the relation that meets the inferred constraint must necessarily meet the supporting constraints. If part of the support comes from constraints on the other relation, though, there will be a reduction in the number of qualifiers. Again, we can make this limitation clear with an example. Suppose the general rule stated above is altered slightly, so that it states that every French ship that carries refined products must be under 60 thousand tons deadweight:

R: (Cargotype = "refined") ∧ (Registry = "France") → (Deadweight < 60)

The constraint on Deadweight can be still be inferred and there is still a reduction in the number of qualifying SHIPS tuples. This is because there may be ships other than French ships whose deadweight is less than 60 thousand tons. But suppose the rule is altered again to state that all French ships are less than 60 thousand tons deadweight, regardless of what they are carrying or of any other factor. Then the rule is:

R: (Registry = "France") → (Deadweight < 60)

and, given query Q, the Deadweight constraint can still be obtained. This time, however, the constraint did not move across the join boundary. No reduction in the number of qualifying SHIPS tuples is obtained, because every tuple of SHIPS with a Deadweight value under 60 already has a Registry value of "France".

From the discussion of constraint movement between joined relations, we conclude that

H4. Move a constraint across a join boundary. *A relation involved in a join to a sufficiently strongly restricted relation is a target for constraints.*

The qualification that the joining relation be "sufficiently strongly restricted" arises for the following reason. If a relation has strong constraints on its attributes and it is to be joined to a relation with very weak constraints on its attributes, then it is very unlikely that a usefully strong constraint can be inferred from it across the join boundary. If the relation it is joined to is not itself restricted, then no constraint can be moved across the boundary.

Another qualification must be added to the preceding heuristic, related to clustering links and to another generalization from section 2.5:

- G6. The cost of a join decreases substantially as the strength of restrictions on the joined relations increases, except on a relation which is clustered with respect to the join term (and is therefore likely to be the "inner" relation of the join method).

Suppose relation X is to be joined to relation Y and that there is a clustering link from X to Y. Then it is extremely likely that a conventional optimizer such as the System R optimizer [Selinger79] will choose to perform the join using X as the outer relation and Y as the inner relation in the manner described earlier. That is, X is scanned and for each qualifying tuple, the pointer (or equivalent index) gives the corresponding tuples of Y quite inexpensively. In this case, no additional constraint on Y can be applied effectively to reduce the cost of the scan, and there is no point in reducing the number of qualifying Y tuples by adding constraints because Y is the inner relation of the join. Hence, Y should not be a constraint target in this case, and we have the additional heuristic:

H5. Don't push a constraint down a hierarchy. *A relation should not be a target for constraints if it is joined to a restricted file from which it has a clustering link or equivalent index.*

From our consideration both of scanning one relation and of joining two relations, we can suggest another heuristic as well:

H6. Use a strongly restricted clustered index. *If a file is strongly constrained on an attribute with a clustered index, then it should not be a target for constraints.*

This heuristic applies whether the relation is the only one in the query or is joined to other relations. In the former case, the relation will be scanned by way of the already constrained attribute. In the latter case, the strong constraint on the indexed attribute makes the relation a likely candidate to be the inner relation, hence reducing the number of qualifiers is not helpful. Besides, the strength of the constraint makes it unlikely that further reductions in the number of qualifiers can be obtained.

Finally, the generation of new constraints makes it possible to render some retrieval operations unnecessary. The target in this case is a query relation that only serves to restrict another relation and from which no information is to be output. If the restrictions on that relation can be found to be superfluous, that is, derivable entirely from constraints on other query relations, then it can be eliminated and the join to it eliminated at a great cost saving. We sum this up as follows:

> **H7. Try to eliminate a dangling relation.** *If a relation is joined to just one other relation and none of its attributes contribute to the output, then it is a target for constraints.*

4.4.1.3 Summary of QUIST's constraint generation heuristics and classes of query transformations

Let us summarize the discussion of QUIST constraint generation heuristics by grouping the heuristics into those that designate constraint targets and those that designate nontargets. The heuristics that designate targets are shown in Figure 4-2. With each of these heuristics, we indicate the kind of query transformation it contemplates, in terms of changes in scanning or joining operations.

- H1. *Try to exploit a clustered index.* Try to obtain a constraint on an attribute of a relation which is restricted in the query and which has a clustered indexed attribute that is not restricted in the query.
  - This heuristic contemplates the replacement of a segment scan by a clustering index scan. We refer to this transformation as ***index introduction.***
- H2. *Push a constraint up a hierarchy.* A relation should be a constraint target if it has a clustering link into a much larger file that is constrained in the query, even if the relation itself is not in the original query.
  - This heuristic contemplates the addition of a join to the query, referred to as ***join introduction.*** The effect of the added join is similar to replacing a segment scan of the linked relation by a clustering index scan of that relation.
- H4. *Move a constraint across a join boundary.* A relation involved in a join to a sufficiently strongly restricted relation is a target for constraints.
  - In this case, the objective is to reduce the number of inner scans of the join by obtaining additional restrictions prior to the cross referencing part of the operation. Hence, the transformation is called ***scan reduction.***
- H7. *Try to eliminate a dangling relation.* If a relation is joined to just one other relation and none of its attributes contribute to the output, then it is a target for constraints.
  - This heuristic is aimed at ***join elimination*** by means of inferring from other query contraints the constraints on the dangling relation specified in the query.

Figure 4-2: Heuristics that designate constraint targets.

In Figure 4-3, we show those constraint generation heuristics that designate relations that are not to be targets for constraints.

- H3. *Don't introduce unlinked joins.* With the exception of the clustering link (hierarchical) case, do not generate constraints for relations that are not part of the original query.

- H5. *Don't push a constraint down a hierarchy.* A relation should not be a target for constraints if it is joined to a restricted file from which it has a clustering link or equivalent index.

- H6. *Use a strongly restricted clustered index.* If a file is strongly constrainted on an attribute with a clustered index, then it should not be a target for constraints.

Figure 4-3: Heuristics that designate nontargets.

4.4.1.4 Constraint targets for the example query

Let us now consider the identification of constraint targets for the example query:

Q: (Deadweight > 400) ∧ (Shiptype = "supertanker")
∧ (Dollarvalue < 1000) ∧ (Facilitytype = "offshore");
(? Destination)

The attributes named in the query reside on three underlying real relations. Attributes are constrained on SHIPS, CARGOES, and PORTS, and an attribute is to be output from CARGOES.

Each of these three relations is designated as a target for constraints by heuristic H4 (move a constraint across a join boundary) because they are all involved in joins with another constrained relation, and because neither of the exceptions in heuristics H5 (don't push a constraint down a hierarchy) or H6 (use a strongly restricted clustered index) apply. Both SHIPS and PORTS are also designated as targets by heuristic H7 (try to eliminate a dangling relation) because both are joined just to CARGOES and neither has an attribute involved in the output.

In addition, the OWNERS relation is designated as a constraint target by heuristic H2 (push a constraint up a hierarchy). The OWNERS file is much smaller than the SHIPS file and there is a clustering link from OWNERS to SHIPS.

Finally, the POLICIES and INSURERS relations are designated as nontargets by heuristic H3 (don't introduce unlinked joins). The inclusion of either of these relations would introduce a costly join.

Now that we have designated appropriate targets for additional constraints, it remains to be seen how to use this information to guide the semantic query transformation process. This issue is taken up in the next section.

4.4.2 Step 2 - Generation of new constraints

We now describe the next step of QUIST's production of semantically equivalent queries: the process of inferring additional constraints on database attributes. QUIST's inference process is based upon the methods of semantic query transformation of general relational calculus queries described in Chapter 3 (see also Section B.4 of Appendix B). We first show how the inference process works on the example query. Following the example, we describe QUIST's general rules for generating new constraints and for merging new constraints with an existing set of constraints. We conclude by noting the conditions under which it is permissible to introduce a constraint on an attribute associated with a relation not previously involved in the query.

4.4.2.1 Selection of rules for the generation of new constraints

The constraint generation step begins with a set of query constraints and with a set of relations designated as constraint targets. A set of rules is then extracted from QUIST's knowledge base. These rules are used to assert new attribute constraints. To be among the rules selected for the assertion of new constraints, a rule must pass several tests:

- *Relevant.* The rule must be *relevant* to the constraints in the query. For a bounding rule, it is necessary that one of its mutually constraining attributes be constrained in the query; we refer to this as the *relevant* attribute. For a production, there are two possible ways to be relevant: either the single attribute constrained on its right hand side is involved in the query, or every attribute constrained on the left hand side is involved in the query. As with a bounding rule, the term relevant attribute (or attributes) is used. If relevance is achieved by means of the right hand side attribute, then one more condition must hold: there must be only one left hand side constraint. The reason for this is to avoid asserting a disjunction; this point is further discussed in Section 4.4.2.2.

    - For our example, rules R4, R5, and R8 (Section 4.4) are eliminated by the relevance test. For all rules but these, one side of the rule entirely involves constraints on Deadweight, Dollarvalue, Shiptype, or Facilitytype. Rules R4 and R8 do not even mention any of the attributes constrained in the query. The right hand side of rule R5 constrains Facilitytype. Therefore, R5 would be relevant except that its left hand side has more than one constraint. Rule R5 is not relevant from its left hand side because although it contains a constraint on Dollarvalue, it also contains a constraint on Cargotype, hence the rule fails the "entirely" part of the relevance test.

- *Promising.* If the rule is relevant, it is then tested to see if it is *promising.* This is a test based on the expected usefulness of the constraint that can be asserted using the rule. A bounding rule involves two attributes. One of them is the relevant attribute; the other is the potential site of the new constraint, which we call the *candidate* attribute. For a production, the attribute on the opposite side from the relevant attribute or attributes is the candidate attribute. A rule is heuristically promising if and only if the candidate attribute is associated with a relation in the list of constraint targets. The point of this test is to avoid long chains of inference that have relatively little likelihood of producing a constraint where we want one.

- The constraint targets are SHIPS, PORTS, CARGOES, and OWNERS. Among the candidate attributes of the relevant rules, Coverage and Issuer are not associated with one of these relations; they are associated with POLICIES. Thus, rules R3 and R7 fail the test of promise: we don't wish to bring in constraints on the POLICIES relation.

- *Applicable.* Every relevant and promising production must be tested to see if it is *applicable.* A production is applicable if and only if each of its relevant attributes is constrained at least as strongly by the query as by the rule itself. Every bounding rule is automatically applicable.

  - Rules R1, R2, and R6 are still possibilities. It turns out that all these rules are applicable. For example, the query constrains the relevant attribute Deadweight with the constraint (Deadweight > 400). Rule R1 constrains Deadweight with the constraint (Deadweight > 350), which is a consequence of the query constraint. Note that rule R3 would have passed the applicability test, but that rule R7 would not because it requires a stricter constraint on Dollarvalue than the one specified by the query.

Every rule that is relevant, promising, and applicable can be used to determine a new constraint on an attribute. The constraint is considered to be *effective* if the result of asserting it in conjunction with the corresponding constraint in the query results in a stronger constraint than the query constraint.

The following new constraints can be asserted:

From R1: (Facilitytype = "offshore")

From R2: (Business = "leasing")

From R6: (Shiptype $\in$ {"supertanker" "carrier"})

The first two of these constraints are effective in that they are at least as strong as the prior constraint on the same attribute. The last constraint is not as strong as the query constraint (Shiptype = "supertanker") so the constraint from R6 is not effective.

If some rules pass the three tests and some effective new constraints are obtained, then a new round of constraint generation begins; otherwise, the constraint generation step ends. In each succeeding round of the constraint generation step, we seek just those rules that were not applicable in any earlier round. For instance, no applicable production is allowed to be used again. Thus, rules R1, R2 and R6 are no longer in consideration after the first round of constraint generation. Furthermore, attributes that have just been more tightly constrained in the last round are distinguished from attributes that were constrained in earlier rounds; the set of relevant attributes in the relevance test for rules must contain at least one newly constrained attribute. In this way, we avoid the repeated retrieval of a rule whose attributes on one side are all constrained but which has already been used to assert a constraint or has been shown to be unpromising or inapplicable. For instance, rule R3 will not be relevant to the second round of constraint generation as Dollarvalue was not newly constrained after the first round. If Dollarvalue receives a stronger constraint in a later round, rule R3 will be relevant again.

We start the second round of constraint generation for the example query. Apart from the eviously described elimination of rules from consideration, the major difference in this round is that rule R8 is now relevant (because the Business attribute was constrained in the first round). However, rule R8 is not applicable because its constraint on Business, (Business = "petroleum"), does not follow from the newly asserted constraint (Business = "leasing"). No other constraints are asserted in this round, so the process of constraint generation terminates.

4.4.2.2 Semantic equivalence transformations in QUIST

We now describe the general conditions related to the generation of constraints in QUIST as the basis for semantic equivalence transformations of QUIST queries. The discussion here appeals to an intuitive notion of inference for the particular kinds of expressions and rules admitted by QUIST. In Appendix B, we show how semantic equivalence transformations in QUIST are actually a special case of such transformations for relational queries, and therefore how the formal definitions advanced in Chapter 3 apply to QUIST as well.

As noted earlier, there are two kinds of rules in QUIST: bounding rules and productions. Once a rule has been selected to try to generate a new constraint, the ensuing manipulations are domain-independent; that is, they depend only upon properties of mathematical and set operators. It should be noted too that the result of any QUIST inference is the assertion of a single constraint on a single attribute.

The conjunctive form of query permitted in QUIST lends itself to a simple form of semantic equivalence transformation. The restriction portion of a QUIST query Q can be represented as

$$Q: C_1(A_1) \wedge C_2(A_2) \wedge \ldots \wedge C_n(A_n).$$

Query Q involves constraints on the set of attributes $\{A_1, A_2, \ldots, A_n\}$, a subset of all the attributes in the virtual relation. Each term $C_i(A_i)$ is one of the constraint forms defined earlier.

Given a conjunctive query Q, the basic semantic transformation operation of QUIST is as follows:

1. Select some semantic rule R from the QUIST semantic knowledge base.
2. If possible, use rule R and query Q to produce a new constraint C'(A) on attribute A.
3. If a new constraint C'(A) is produced, combine it with Q to form the transformed query Q'.

We have already discussed QUIST's rule selection tests and the heuristics that they use. Here we assume that a rule R has been selected and we discuss how a new constraint C'(A) can be produced. In Section 4.4.2.3, we discuss how the new constraint can be merged with query Q to form the semantically equivalent transformed query Q'.

We examine this first in the context of a bounding rule. A QUIST bounding rule is of the form:

$$A_1 \theta A_2$$

where A_1 and A_2 are numerical-valued database attributes and θ is a standard Boolean comparison operator such as less-than or greater-than. The bounding rule places an upper bound on one of the attributes and a corresponding lower bound on the other (except in cases of equality and inequality). If a query constrains one of the attributes by, say, placing an upper bound on it, and if the bounding rule indicates that the constrained attribute serves as an upper bound for the other attribute, then that other attribute inherits the same upper bound. Similar remarks hold for a lower bound.

As an example, consider bounding rule R4 from the example knowledge base. It states that a ship carries no more cargo than its rated capacity, and is represented to QUIST as (Quantity ≤ Capacity). It is natural to think of the value of Capacity as providing an upper bound on the value of Quantity; it is equally correct to think of the value of Quantity providing a lower bound on the value of Capacity. Suppose a query contains the constraint (Quantity > 100); that is, the query places a lower bound on Quantity. Then it is easy to see that a lower bound constraint on Capacity, (Capacity > 100), can be inferred. In a similar way, a query with a constraint (Capacity < 250) permits the inference of the constraint (Quantity < 250). If the query instead contains the constraint (Quantity < 100), then it is not possible to use the example rule to infer anything about Capacity, and similarly for the constraint (Capacity > 300).

Turning now to productions, we will see that constraint generation draws on properties of both numerical and set operators. As noted earlier, a QUIST production is a rule of the form

$$C_1(A_1) \wedge \ldots \wedge C_n(A_n) \rightarrow C'(A')$$

where each term $C_i(A_i)$ signifies a constraint on some database attribute and where a given attribute appears at most once on the left hand side.

With QUIST productions, it is possible to reason left-to-right or right-to-left. In reasoning left-to-right, it is necessary to show that the query constrains all the attributes on the left hand side of the rule, and that every such rule constraint follows from the corresponding query constraint by the properties of numerical or set comparison.† If so, then the rule's right hand side constraint can be asserted.

Reasoning right-to-left is limited to productions with a single constraint on the left hand side. This is because right-to-left reasoning deals with the contrapositive of the rule. The negation of a multiterm conjunction is a multiterm disjunction, but QUIST, in common with many other inference systems, makes no inferences with disjunctions of terms. In this mode of reasoning, if it is seen that the negation of the right hand term follows from the corresponding query constraint, then the negation of the left hand term can be asserted. Obtaining the negation of a constraint on a string-valued attribute is simply a matter of exchanging $\notin$ for $\in$, or vice versa. For constraints on numerical-valued attributes, it is a matter of "inverting" the interval specified in the constraint. For instance, the negation of the constraint ($\wedge$ge $\in$ ((18 65])) is ($\wedge$ge $\in$ ((-∞ 18] (65 ∞))).

---

†In particular, QUIST does not set up inference subgoals.

To illustrate inference with QUIST productions, consider a production that states that juice and bananas are always carried in refrigerated ships:

(Cargotype ∈ {"juice" "bananas"}) → (Shiptype = "refrigerated").

Suppose the query contains the restriction (Cargotype = "bananas"). The rule's constraint on Cargotype follows from this corresponding query constraint because the set of values permitted in the query is a subset of the values permitted by the rule. Therefore, the constraint (Shiptype = "refrigerated") can be asserted. If, on the other hand, the query contains the constraint (Shiptype = "supertanker"), then the production given above can be used to assert the constraint

(Cargotype ∉ {"juice" "bananas"}).

4.4.2.3 Merging a new constraint with an existing query

We have seen how QUIST generates additional constraints on attributes. In this section, we describe in general how new constraints are combined with an existing QUIST query.

The result of any one of the inference processes just described is the assertion of a single new constraint C′on an attribute A. The processes can be viewed as follows: given some conjunction T of terms $C_i(A_i)$, it is possible to infer a new term C′(A); that is, T → C′. From this point of view, combining the new constraint with the old ones follows along the lines described in Section 3.6 on logical transformations in semantic query optimization, with some additional factors arising from QUIST's joinless representation and from the task of detecting unsatisfiable query constraints.

To be more specific, let query Q be represented as before:

Q: $C_1(A_1) \wedge C_2(A_2) \wedge \ldots \wedge C_n(A_n)$.

By the logical equivalence

$(A \wedge (A \rightarrow B)) \equiv (A \wedge B)$

query Q can be transformed† into the semantically equivalent query:

Q′: Q ∧ C′(A)

The new query Q′is actually formed by replacing the prior constraint C(A) on attribute A by the conjunction of C(A) and the new constraint C′(A). The resultant constraint is obtained as follows:

1. If there is no prior constraint C(A), then the resultant constraint is merely C′(A).
2. If the prior constraint C(A) is stronger than C′(A), then the resultant constraint remains C(A).

---

†In most cases; but see Section 4.4.3.1.

3. If the new constraint C′(A) is as strong or stronger than C(A), then the resultant constraint is C′(A).

4. If C(A) and C′(A) overlap in the sense that C permits some values of A that C′does not permit and vice versa, then the resultant constraint is the intersection of the values they permit. For instance, if C(A) is (A ∈ {"a" "b"}) and C′(A) is (A ∈ {"b" "c"}), then the resultant constraint is (A = "b"). An analogous combining rule is observed for numerical interval constraints.

5. Finally, if C(A) and C′(A) conflict in the sense that there are no values of A that can satisfy both constraints, then the original query restrictions are not satisfiable in the database; that is, the answer to the originial query must be the empty set. Note that this is detected without recourse to the data.

4.4.3 Step 3 - Formulation of the set of semantically equivalent queries

We now discuss the last step of QUIST's generation phase: the formulation of a set of alternative, semantically equivalent queries from the constraints generated in the preceding step. A simplified way to look at the final step of query formulation is as follows. After constraint generation, there is a set of constraints on database attributes. Some of these constraints must be part of the query while other constraints are optional. A constraint is optional if it can be derived from other query constraints. One of the queries in the set of semantically equivalent queries is a "kernel" query, Q_0, that includes only the necessary constraints. If no new constraints are generated, then Q_0 is the original query. If there are N additional optional constraints, then the set of equivalent queries includes an additional 2^N-1 queries generated by all possible choices of including or excluding those N constraints.

This account must be modified in several ways. First, it is not always possible to classify every constraint as necessary or optional independently of the classification of the other constraints. What may happen is that two sets of constraints are related, so that one set or the other may be excluded, but not both. Second, the addition to the query of certain derivable constraints may implicitly introduce new relations into the query. Introduction of new relations is only permitted if the database meets certain structural constraints. Finally, QUIST assumes that once a particular (real) relation is involved in a query, every constraint on attributes of that relation should be part of the query. The reason is that additional constraints on a relation cannot increase the cost of processing, given QUIST's cost measure, the number of page fetches from secondary storage. Therefore, the number of independently excludable constraints is reduced.

In the remainder of this section, we indicate how the set of equivalent queries is formulated for our example. After that, we give the details about when constraints can be considered optional, and when new relations can be introduced.

We started with the QUIST query

Q: (Deadweight > 400) ∧ (Shiptype = "supertanker")
∧ (Dollarvalue < 1000) ∧ (Facilitytype = "offshore");
(? Destination)

and the constraints generation step left us with the following constraints:

(Deadweight > 400), (Shiptype = "supertanker"), (Dollarvalue < 1000),
(Facilitytype = "offshore"), (Business = "leasing").

The only new constraint derived in that step is (Business = "leasing"), although it is now known that the Facilitytype constraint is derivable from other query constraints, namely from the constraint on Deadweight using rule R2.

Let us denote the five constraints by C_1 through C_5 as follows:

C_1: (Deadweight > 400)

C_2: (Shiptype = "supertanker")

C_3: (Dollarvalue < 1000)

C_4: (Facilitytype = "offshore")

C_5: (Business = "leasing")

Then C_1 through C_3 are the necessary constraints and C_4 and C_5 are the optional constraints. The kernel query Q_0 contains just the necessary constraints:

$$Q_0 \equiv C_1 \wedge C_2 \wedge C_3$$

which, incidentally, is *not* the original query because the constraint on Facilitytype has been identified as optional. There are two optional constraints, each on a separate underlying relation, so there are three other equivalent queries:

$Q_1 \equiv Q_0 \wedge C_4$ (the original query)

$Q_2 \equiv Q_0 \wedge C_5$

$Q_3 \equiv Q_0 \wedge C_4 \wedge C_5$

The cost of the alternative queries can be estimated by determining the real relations they involve. The kernel query Q_0 involves attributes on SHIPS and CARGOES; that is, it is possible not to involve PORTS at all, because PORTS is not involved in the output and the only constraint on one of its attributes is derivable from constraints on other relations. All the other queries involve SHIPS and CARGOES, while bringing in additional relations: Q_1 adds PORTS (Q_1 corresponds to the original query), Q_2 adds OWNERS, and Q_3 brings in both PORTS and OWNERS.

The OWNERS relation is of course not involved in the original query. In order for queries Q_2 and Q_3 to be equivalent to the original query, it is necessary that every tuple in SHIPS have a corresponding tuple in OWNERS. If this condition is not met, then it is possible that some tuples in SHIPS that satisfy the original query conditions will not satisfy the join condition.

QUIST has now generated four equivalent queries, each involving a different set of database relations. This concludes the first phase of the QUIST system. The next phase is to determine which of these queries has the lowest estimated processing cost. Before we discuss this, we discuss the conditions under which relations can be added to a query (as OWNERS is added to get queries Q_2 and Q_3), and the conditions under which relations can be dropped from a query (as PORTS is dropped from the original query to get queries Q_0 and Q_2).

4.4.3.1 The introduction of joins

It was noted earlier how the query $Q \wedge C'(A)$ is semantically equivalent to query Q in most cases, if constraint $C'(A)$ can be derived from the other constraints in Q. The possible exception arises when the addition of $C'(A)$ implicitly introduces a new (real) relation into the query, the relation R to which attribute A is associated. The introduction of a new relation also introduces one or more joins to connect R with the real relations already involved in Q.

This section discusses the conditions under which it is permissible to introduce new relations and new joins into a query. Briefly, it is only all right to do so if no tuples in the original query fail to satisfy the join terms that must be introduced. We illustrate this idea with an example here. It is discussed more completely in Appendix B.

An example illustrates the introduction of joins. Suppose the query requests the destination of all cargoes of refined petroleum products:

Q: (Cargotype = "refined"); (? Destination)

and suppose it is known that refined petroleum products are only carried by ships whose deadweight does not exceed 60 thousand tons:

R: (Cargotype = "refined") → (Deadweight ≤ 60).

Cargotype is associated with the CARGOES relation, and Deadweight is associated with the SHIPS relation that is not involved in the original query Q. The straightforward query transformation produces a request for the destination of all cargoes of refined petroleum products that are being carried by ships of under 60 thousand tons deadweight:

Q': (Cargotype = "refined") ∧ (Deadweight ≤ 60); (? Destination)

The new query Q' implicitly introduces a join between CARGOES and SHIPS. One way to process Q' is to find all ships not exceeding 60 thousands tons deadweight, then to find the cargoes they are carrying and indicate the destinations for the cargoes that are refined petroleum products. However, consider some tuple x in the CARGOES relation. If the Ship attribute of x has a null value, or if it contains the name of a ship that is not listed among the tuples of the SHIPS relation, then the join will miss tuple x, even though a simple scan of CARGOES requested by the original query Q will return tuple x.

The difficulty is related to the *structural semantics* [ElMasri80b] of the database. The fact that

there is no value for the Ship attribute of a tuple in CARGOES may be a data entry oversight, or it may reflect a database design decision to permit null values and thus to interpret a cargo as existing independently of any ship that may carry it. The fact that the latter interpretation results in a null value in some field is an artifact of the manner in which relationships can be represented in the relational model. The former case, in which the value in the Ship attribute of some tuple in CARGOES does not correspond to the value in the Shipname attribute of any tuple in SHIPS, is much more likely to be an error in data entry.

In any event, query Q' is semantically equivalent to query Q if and only if for every tuple in CARGOES there exists a corresponding tuple in SHIPS, where "corresponding" refers to tuples in SHIPS that would be logically accessed from CARGOES by way of the logical access path defined for QUIST's virtual relation.

It is assumed throughout the QUIST system that the appropriate structural constraints on the existence of corresponding tuples are enforced, so that the introduction of joins is always permitted.† The system could be modified to make this assumption unnecessary. It would be necessary to incorporate another test to see if the existence condition does in fact hold when the introduction of a particular join is considered.

4.4.3.2 The elimination of query constraints

As pointed out in Section 3.6, it is possible not only to add constraints to a query, but also to eliminate constraints if they are derivable from other constraints in the query. A constraint on a single attribute can be eliminated despite the fact that it was constrained in the original query, provided that another equally strong or stronger constraint can be derived on the same attribute based solely on initial constraints on other attributes. Similar conditions hold for the elimination of constraints on more than one originally constrained attribute, although care must be taken to avoid eliminating sets of constraints that support each other's derivation.

The following example illustrates the possible pitfall in constraint elimination. Let query Q contain the constraints $(A_1 > 10) \wedge (A_2 > 30) \wedge (A_3 > 50)$. Assume there are two production rules, R_1 and R_2:

R_1: $(A_1 > 5) \wedge (A_3 > 10) \rightarrow (A_2 > 40)$

R_2: $(A_2 > 25) \wedge (A_3 > 40) \rightarrow (A_1 > 20)$

With rule R_1, it is possible to infer the new constraint $(A_2 > 40)$, with constraints on A_1 and A_3 in its basis. With rule R_2, we obtain $(A_1 > 20)$, whose basis includes constraints on A_2 and A_3. Hence, both A_1 and A_2 are candidates for constraint elimination. Yet if both are dropped, yielding the query $(A_3 > 50)$, then there is no guarantee that the items retrieved by that query satisfy the constraints on

---

†This condition is made more precise in Appendix B.

either A_1 or A_2. The problem is that the derived constraint on either attribute requires a constraint on the other one.

The details of the analysis of constraint elimination in QUIST are as follows. Suppose that query Q'has been formed from the original query Q through several steps of inference and merging, and that Q'constrains attributes A_1 through A_n. For every attribute A in this set, one of three conditions holds:

1. Attribute A was not constrained by the original query Q. Clearly, then, the constraint C'(A) in Q'is not essential for obtaining the desired answer and can be eliminated.

2. Attribute A was constrained by the original query Q and no other, stronger constraints have been derived on it. Therefore, constraint C'(A) in Q'is essential and must not be eliminated.

3. Attribute A was constrained by the original query Q but other, stronger constraints have been obtained on A during the inference and merging that produced the constraint C'(A) in Q'.

Thus, constraints on all attributes in class 1 can be eliminated, but no constraints on attributes in class 2 can be eliminated. Whether or not constraints on class 3 attributes can be eliminated depends upon what is called the *basis* of the contraints. This explicit maintenance and use of inference dependencies to reason about the necessity of constraints is akin to the set-of-support ideas for derived information used for "truth maintenance" systems ([Fikes75], [Doyle78]).

The basis of a constraint C on some database attribute A is defined to be the set of constraints in the original query which must hold in order for C to hold. Before any steps of inference and merging, the basis of each constraint imposed in the initial query contains just the constraint itself. Let Q'be the current query, and let $\{C_1(A_1), \ldots, C_K(A_K)\}$ be the set of constraints in Q'that enable constraint C'(A) to be asserted using some semantic rule R. The basis of C'(A) is the union of the bases of those constraints. C'(A) is now merged with Q according to the rules listed in the preceding section. When C'(A) is strictly stronger or weaker than the prior constraint C(A), the basis of the resultant constraint is simply the basis of the stronger constraint. When the two constraints overlap, the basis of the resultant constraint is the union of the bases of the new and the prior constraints.

To return to the question of eliminating constraints, first consider individually each class 3 attribute, that is, each attribute that is constrained in the original query Q and upon which additional constraints have been obtained. Let C(A) be the constraint on A in the initial query Q, and let C'(A) be the constraint on A in transformed query Q'. Constraint C'(A) on transformed query Q'can be eliminated if and only if constraint C(A) from original query Q is not in the basis of C'(A); in other words, only if C'(A) is derivable entirely from query constraints on attributes other than A.

When considering the elimination of constraints from several attributes that are constrained in the original query, it is necessary to avoid eliminating too many constraints, as the earlier example illustrated. That situation is avoided by retaining the rule against eliminating a constraint on an

attribute that appears in its own basis, and employing a procedure to keep track of the ultimate basis of a constraint when other constraints are eliminated. Suppose there are several candidates for constraint elimination. Let the constraint on attribute A meet the test for single constraint elimination. Suppose that constraints on attributes B and C are in the basis of the constraint on A. When the constraint on A is eliminated, it should be dropped from the bases of all other constraints and replaced by the constraints on B and C (unless they already appear).

In the example, suppose we choose to eliminate the constraint on attribute A_1. Its basis contains constraints on A_2 and A_3. The constraint on A_2 does contain the constraint on A_1, so the constraint on A_1 replaced by the constraints on A_2 and A_3. Since A_3 already appears in the basis, the new basis consists of constraints on A_2 and A_3. But now, A_2 no longer meets the test for constraint elimination, because it appears in its own basis. Thus, we are left with the equivalent query $(A_2 > 40) \wedge (A_3 > 50)$. It is easy to see by production rule R_2 that items that satisfy this query also satisfy the original constraint $(A_1 > 10)$.

4.4.4 Step 4 - Determing the lowest cost query

The last task of the QUIST system is to take the set of queries produced in the preceding steps and to estimate which one costs the least to carry out. In a sense, this process is not an integral part of semantic query optimization because it is merely a matter of performing a conventional query optimization analysis for a set of queries, rather than for a single query.

QUIST's query cost estimator is derived from the one described for System R [Selinger79]. The assumptions behind the System R query optimization were reviewed in Chapter 2. QUIST's cost estimator differs from that of System R chiefly in assuming that a join is carried out by the nested loops method rather than choosing between that method and the merging scans method. This is a reasonable simplification and does not affect the heuristics; for instance, it would still make sense to move constraints across join boundaries prior to performing sorts. Another difference from the System R optimizer is that QUIST's cost measure involves just the number of estimated page fetches rather than combining this with an estimate of CPU activity. However, results reported in [Astrahan80a] suggest that the number of page fetches is a suitable cost measure for the class of queries admitted by QUIST.

QUIST's estimator differs from System R's in one other respect: the estimation of restriction selectivities (Section 4.4.1.1). System R assumes that all constraints are independent, hence the estimated selectivity of a conjunction of constraints is the product of the estimated selectivity of each constraint alone. On the other hand, the QUIST estimator must distinguish between given and derived constraints. A derived constraint is, of course, not independent from the constraints from which it is derived. In QUIST, the estimated selectivity of the conjunction of a given constraint and a constraint derived from it is taken to be the estimated selectivity of the given constraint alone.

We have now concluded the description of how QUIST operates. In the next chapter, we discuss the system's effectiveness.

Chapter 5
The effectiveness of the QUIST system

QUIST has been implemented to investigate the design of effective systems for semantic query optimization. We described issues in the design of such effective systems in Section 4.1. In this chapter, we address the question of QUIST's effectiveness from several different perspectives.

In Section 5.1, we present details of the cost model used in QUIST's cost estimation step (the last step). We use this cost model to provide quantitative estimates of the reduction in the cost of processing that is obtained by effecting each of the transformations defined in Section 4.4.1.3: *index introduction, join introduction, scan reduction,* and *join elimination.*

We then examine timing results for a range of queries in Section 5.2. Processing of the selected queries illustrates each of the indicated transformations, as well as the ability of QUIST to decide that no inference is apt to be fruitful or to recognize when the original query restrictions cannot be satisfied.

Finally, we take up the question in Section 5.3 of the continued effectiveness of QUIST's control strategy as the size of the database or the number of semantic rules increases.

5.1 Quantitative estimates of query improvements

This section presents estimates of the quantitative improvement that can be obtained for each of the four QUIST transformations: index introduction, join introduction, join elimination, and scan reduction. It is possible that the application of one of these transformations results in a complete change in the sequence of processing the complete query. Hence, it is not possible to state directly what the overall effect on query cost will be of any given transformation. Instead, estimates are presented for local changes, as if the transformation had no other effect. Additional changes at the resequencing level would lower costs even further.

5.1.1 Processing assumptions and cost formulas

First, we briefly review our assumptions concerning how scans and joins are performed. Each relation is assumed to reside on a single file that is divided into pages of P records each, where P is a systemwide constant. Furthermore, it is assumed that each file entirely fills the storage segment in which it resides. The effect of this assumption is that the cost to perform a sequential segment scan is simply the number of pages in the file, because we read no pages associated with other files. This assumption leads to underestimates of the improvements brought about by transformations that eliminate segment scans.

A join is performed on two relations; throughout this section,we assume that R_1 is the outer relation and R_2 is the inner relation, unless stated otherwise. R_1 is scanned by means of a sequential scan or an indexed scan. We only consider clustering indexes in this section. For every qualifying tuple in R_1, we find the corresponding tuples in R_2. This is achieved either by a sequential scan of R_2, or by a clustering indexed scan if we have a constraint on an indexed attribute of R_2 other than the join attribute, or by what can be called a link scan, a clustering indexed scan in which the clustering index is on the join attribute. The cost to find the corresponding records varies according to the method of the inner scan.

Based on the processing assumptions, we develop necessary cost formulas. Let N_i be the number of records in the file that corresponds to relation R_i. Therefore, the file occupies N_i/P pages, and the cost $S(R_i)$ to perform the sequential segement scan is given by

$$S(R_i) = N_i/P.$$

The cost of a join depends on the number of qualifying items in the outer relation. This in turn depends upon the selectivity of the restrictions on that relation. Let α_i be the selectivity value of the restrictions on relation R_i, where $0 \le \alpha_i \le 1$. Then the number of qualifying tuples from relation R_i is $\alpha_i N_i$.

Unless otherwise stated, we assume that the outer relation of a join is scanned by means of a sequential segment scan. Given that assumption, there are three cost formulas for the join between relation R_1 and R_2. The appropriate formula for the cost $J(R_1,R_2)$ depends upon how tuples of R_2 are found to match the current qualifying tuple of R_1. If R_2 is scanned by means of a sequential scan, the cost is

$$J(R_1,R_2) = N_1/P + \alpha_1 N_1 N_2/P.$$

If R_2 is scanned by means of an indexed scan on a clustering index of an attribute other than the join attribute, then the join cost is given by

$$J(R_1,R_2) = N_1/P + \alpha_1 N_1 \alpha_2 N_2/P.$$

The difference between these two costs is due solely to the fact that only $\alpha_2 N_2/P$ pages need to be scanned for each of the $\alpha_1 N_1$ scans of R_2 for an indexed scan, versus N_2/P pages each time for a sequential scan. The figure for an indexed scan relies on the assumption that the values permitted by

the restriction on the indexed attribute are clustered together rather than scattered throughout the relation. This is the case, for instance, when a numerical attribute is confined to some interval.

If there is a clustering index on the join attribute of R_2, we assume that the query processor or the underlying file system is sophisticated enough to maintain its "current place" in the scan of R_2 within an in-core buffer, so any page of R_2 is read at most once. In other words, rather than scanning R_2 completely for every qualifier in R_1, the system first checks to see if the proper page is in the buffer. The next page of R_2 is brought in from the disk only if it is not in the buffer or if the end of the previous page is reached while reading matching tuples. In this case, then, the cost to perform the join is merely the cost to scan R_1 plus the number of pages of R_2 that hold tuples that correspond to R_1 qualifiers.

To determine the fraction of pages with corresponding tuples, we make the additional assumption that there is a 1 to N relationship between records of file R_1 and file R_2. This seems a reasonable assumption for the kind of hierarchical link that is implemented by the index just described. Under this assumption, every tuple in R_1 has, on the average, N_2/N_1 corresponding tuples in R_2. We refer to the inverse of this as β, so that $0 \leq \beta \leq 1$. The fraction of pages of R_2 which are brought in is approximately the same fraction of pages on which there are qualifiers in R_1. If we assume that those qualifiers are bunched and not randomly scattered throughout R_1, then the fraction of R_2 pages brought in is simply α_1. Therefore, we have the following formula for the join with an indexed scan on the join attribute:

$$J(R_1,R_2) = N_1/P + \alpha_1 N_2/P.$$

The considerable saving of this method (the elimination of the N_1 factor in the second term) is due to the clustering of the two files with respect to each other.

5.1.2 Cost improvements from transformations

The join cost formulas are used to show how different QUIST query transformations reduce the cost of processing in selected examples.

5.1.2.1 Index introduction

In index introduction, a constraint is obtained on an index that was not previously constrained. Assume in this example that the index is not on the join attribute. R_1 and R_2 have constraints with selectivites α_1 and α_2, as usual. Suppose a new constraint is inferred on a clustering index of R_1, and assume it depends at least partly on other constraints on R_1 so that the overall selectivity is still α_1. If we keep R_1 as the outer relation, then the cost of the join is

$$J(R_1,R_2) = \alpha_1 N_1/P + \alpha_1 N_1 N_2/P.$$

The original cost is

$$J(R_1,R_2) = N_1/P + \alpha_1 N_1 N_2/P.$$

However, for large files (large values of N_1 and N_2) we expect the cross product terms (the ones that involve N_1N_2) to dominate the costs, so there is only a marginal improvement. However, if there is a constraint inferred on a clustering index of R_2, then the new cost is

$$J(R_1,R_2) = N_1/P + \alpha_1 N_1 \alpha_2 N_2/P$$

assuming that the new constraint does not change the overall selectivity of constraints on R_2. Considering just the cross product terms, if C is the original cost and C′ is the cost after the transformation, then the two costs are related by

$$C' \approx \alpha_2 C$$

where α_2 is a fraction between 0 and 1, hence there could be a substantial reduction in cost.

5.1.2.2 Join introduction

Suppose R_2 is part of the original query and R_1 is not, but there is a clustering link from R_1 to R_2; that is, R_2 has a clustering index on an attribute to which it is joined to R_1, and R_1 and R_2 are in proper sequence with respect to their respective joining attributes. If we infer a constraint on R_1 (and if suitable structural constraints are met -- see Appendix B) then a join between R_1 and R_2 can be introduced.

Consider a case where the original query includes a join between R_2 and some relation R_3. Assuming that neither relation is constrained on an indexed attribute, the cost of performing the join is

$$J(R_2,R_3) = N_2/P + \alpha_2 N_2 N_3/P.$$

if R_2 is the outer relation. The first term is the scan of R_2 and the second term is the cross matching term of R_2 and R_3.

Now suppose R_1 enters through join introduction. There are now two joins to be done. The cost of the join between R_1 and R_2 is given by

$$J(R_1,R_2) = N_1/P + \alpha_1 N_2/P.$$

if R_1 is the outer relation. The two joins can be cascaded so that the cost to join R_2 and R_3 only includes the previous cross matching term. Therefore, the total cost is now

$$C' = N_1/P + \alpha_1 N_2/P + \alpha_2 N_2 N_3/P.$$

The factor in the final cross matching term is assumed to be the same after join introduction as before; that is, it is assumed there will be as many qualifiers from R_2, hence as many scans of R_3. This is based on the assumption that the constraints on R_1 are inferred from constraints already on R_2. If we denote the original cost formula as $C = J(R_2,R_3)$ and if we recall the ratio between the file sizes as $\beta = N_1/N_2$, then we find that the original cost C is related to the new cost C′ by

$$C' = C - [1 - (\alpha_1 + \beta)]N_2/P.$$

If R_1 is the same size as R_2, so that $\beta = 1$, then join introduction clearly is not worthwhile. But if both α_1 and β are near zero, then join introduction can be quite useful.

5.1.2.3 Scan reduction

In this transformation, constraints are inferred "across the join boundary". The point very simply is to reduce the number of qualifiers in the outer scan. If a constraint of selectivity α' is inferred on R_1, and if it is independent of the other constraints already on R_1, then by considering the dominant product term as we did for index introduction we find that the new cost C' is related to the old cost C by the same relationship:

$$C' \approx \alpha C.$$

5.1.2.4 Join elimination

A relation is only involved in the query to constrain another relation; none of its attributes are desired as output and it is only joined to that one other relation. In this transformation, the constraints on the "dangling" relation are shown to be derivable from other constraints in the query, so the join to that relation is simply eliminated.

Generally speaking, this should lead to a reduction in the cost of the query by about the amount needed to perform the join. Therefore, because a join is often very expensive, join elimination may bring about a substantial cost reduction. However, in the case of a clustering link such as supports the join introduction transformation, the elimination of a join may actually increase the cost of the query, so join elimination would not then be desirable.

5.2 Experiments with the QUIST system

To demonstrate the effect of semantic query optimization on the cost of processing queries, a selection of QUIST queries, including the example query of Chapter 4, has undergone semantic query optimization with the QUIST system. The queries are specifically chosen to illustrate various transformations that can be obtained by means of semantic query optimization, and to illustrate the magnitude of the resulting reductions in query processing cost. The query processing cost estimates are based on the model of query processing described in Section 5.1 and depend also upon the assumed size of the files indicated below. The stated time to perform the analysis itself comes from actual measurements, but depends upon the implementation of QUIST.

What these results suggest about the potential importance of semantic query optimization is more significant than the specific numbers reported here. Beyond the particular processing estimates, the results support the contention that semantic query optimization can bring about significant

reductions in the cost of processing queries with an acceptable overhead for analysis. They also suggest the effectiveness of the inference-guiding heuristics.

We assume the same database as the one described in Section 4.4. Physical database parameters have been chosen in order to give some idea of processing time for a moderately large database. We assume that each relation in the example database corresponds to a single physical file with the following number of records:

File	Size
SHIPS	20,000
CARGOES	25,000
PORTS	2,000
OWNERS	200
POLICIES	25,000
INSURERS	500

Table 5-1: Assumed File Sizes for Timing Experiments

We assume that there are twenty records per file page (the same value used in [Yao79]) and that the time per page fetch is thirty milliseconds (extrapolated from [Gotlieb75]). We further assume that the SHIPS file is physically clustered with respect to the OWNERS file.

The example rule base contains approximately thirty rules like the ones in Section 4.4. These rules were obtained in part from Couper's "Geography of Sea Transport" [Couper72].

The QUIST system is implemented in Interlisp [Teitelman78] on a Digital Equipment Corporation DECSystem20 model 60 computer.

5.2.1 Analysis of individual queries

We start with the example query of Chapter 4:

> ***"List the destination of cargoes worth less than one million dollars being carried by supertankers over 400 thousand tons deadweight to ports with offshore load/discharge facilities."***

As indicated in Section 4.4.3, three semantically equivalent queries can be generated in addition to

the original one. These include two useful transformations of the original query: join introduction and join elimination. First, it is worthwhile to add the OWNERS file into the query because SHIPS is much larger than OWNERS and is physically clustered with respect to it. Second, it is possible to show that the constraint on PORTS is superfluous. Because PORTS is only joined to one file and is not involved in the output, it can be eliminated. Hence, the lowest cost transformed query turns out to be:

> ***"List the destination of cargoes worth less than one million dollars being carried by supertankers over 400 thousand tons deadweight owned by leasing companies."***

QUIST's conventional query optimization subsystem determines that for the assumed file sizes and auxiliary structures, the original query in SHIPS, CARGOES, and PORTS can be processed in 435 seconds. The query in OWNERS, SHIPS, and CARGOES that results from join introduction and join elimination can be processed in just 19 seconds. The total time to perform this analysis is 2.1 seconds.

The reduction in cost of well over an order of magnitude for this example query comes from the simultaneous occurrence of fortunate circumstances in the query, database structure, and semantic rules. Indeed, the query was specifically chosen to show what can happen when circumstances are right.

Things are not always so well-suited for semantic query optimization, but there are many situations in which significant improvements can be obtained. We now present a set of queries to illustrate the specific transformations of QUIST: index introduction, join elimination, scan reduction, and join introduction. Other queries in the set illustrate two other important characteristics of QUIST. First, QUIST can detect a query whose qualification cannot be satisfied because of semantic integrity constraints (and which is therefore a null query). Second, QUIST can determine rapidly when there are no opportunities for cost reduction via semantically based transformations.

For this set of queries, it is assumed that the database has clustering indexes on the Shiptype field of SHIPS, the Ship field of CARGOES, the Country field of PORTS, and the Issuer field of POLICIES. The queries are presented along with the rule that is relevant to the particular transformation, the transformed version of the query, if any, and the resulting change in processing, if any.

1. **Index introduction.**

   Query Q1: "List the owners of all ships with a deadweight greater than 200 thousand tons."

   Relevant rule: "Any ship over 150 thousand tons deadweight is a supertanker." (This is a change from example rule R6).

   Transformed query: "List the owners of all *supertankers* with a deadweight greater than 200 thousand tons."

Result: A new constraint on the indexed Shiptype attribute of SHIPS.

2. **Join elimination.**

Query Q2: "List the shipper and quantity of liquefied natural gas cargoes carried by pressurized tankers to Marseilles."

Relevant rule: "Liquefied natural gas is always carried by pressurized tankers."

Transformed query: "List the shipper and quantity of liquefied natural gas cargoes carried to Marseilles."

Result: Elimination of the join with SHIPS because Shiptype constraint is superfluous.

3. **Scan reduction.**

Query Q3: "List the owners and the quantity of cargo of ships carrying refined petroleum products to Danish ports."

Relevant rule: "Refined petroleum products are carried by ships with deadweight under 60 thousand tons."

Transformed query: "List the owners and the quantity of cargo of ships *with deadweight under 60 thousand tons* carrying refined petroleum products to Danish ports."

Result: Constraint on Deadweight can be applied prior to cross matching step of join between SHIPS and CARGOES reducing the number of qualifying SHIPS tuples and therefore the number of scans of CARGOES.

4. **Join introduction.**

Query Q4: "List the owners of supertankers with deadweight over 350 thousand tons that are carrying cargoes to French ports."

Relevant rule: "Only leasing companies own vessels that exceed 300 thousand tons deadweight."

Transformed query: "List the *leasing company* owners of supertankers with deadweight over 350 thousand tons that are carrying cargoes to French ports."

Result: Addition of join between OWNERS and SHIPS has the effect of a more efficient scan of SHIPS.

5. **Detection of unsatisfiable conditions.**

Query Q5: "List the owners of all bulk cargo ships with deadweight over 200 thousand tons."

Relevant rule: "Any ship over 150 thousand tons deadweight is a supertanker."

Result: No transformed query. QUIST indicates that no items can satisfy the query conditions.

6. **Absence of opportunities for cost reducing semantic transformation.**

Query Q6: "List the owners of all refrigerated ships."

Result: Indexed attribute Shiptype is already constrained. No relevant rule. Query remains the same.

We now show the estimated processing times associated with these queries on the example database (Table 5-2).

Query	Transformation	Est. Proc. Time without SQO (sec.)	Est. Proc. Time with SQO (sec.)
Q0	join introduction & join elimination	435	19
Q1	index introduction	30	4
Q2	join elimination	313	37
Q3	scan reduction	1125	519
Q4	join introduction	348	112
Q5	(unsatisfiable)†	30	0
Q6	(SQO deemed useless)	3	3

Table 5-2: Reduction in Processing Costs with SQO

Two times are shown. The first one is the estimated time to process the original query optimized only by conventional means. The second one is the estimated time to process the transformed query, that is, with both semantic and conventional optimization. Again, what is significant is the relative magnitude of processing times rather than the precise times indicated. Note that the amount of processing time for QUIST itself is about 1 second in each case. In the case of the detection of unsatisfiable conditions, the time for QUIST analysis is under half a second. These QUIST analysis times include all four steps described in Chapter 4, including the time it takes to carry out

† The query is a null query. QUIST would not send it to the database for processing.

conventional query optimization for each alternative query. The example query of Chapter 4 is referred to as Q0.

5.2.2 The effect of inference-guiding heuristics

Another important factor in QUIST's effectiveness is the effect of the inference-guiding heuristics. Table 5-3 indicates that QUIST spends much more time generating constraints when the heuristics are not enforced. This effect is compounded because when more constraints are generated, more alternative queries may be formulated, and these additional queries must each undergo conventional optimization.

Query	QUIST -- all steps		QUIST -- inference only	
	time with pruning	time without pruning	time with pruning	time without pruning
Q1	.43 sec.	6.7	.29	1.20
Q2	.90	2.6	.42	.58
Q3	1.00	1.1	.45	.51
Q4	1.30	20.1	.22	.98

Table 5-3: Effect of Inference-Guiding Heuristics

For each of the four queries noted here, four timing figures are given. First there are two timings for all steps of QUIST, with and without pruning based upon heuristics; that is, with and without the use of constraint targets. Second, there are two timings for just the inference portion of QUIST (step 2 described in Chapter 4) with and without pruning. A larger difference is seen when we look at all steps of QUIST rather than just at QUIST's inference steps. As noted above, this reflects the effort to estimate the cost of more alternative queries.

The effectiveness of inference guiding heuristics is suggested by the number of rules tested in the analysis of a query like query Q1 with and without pruning. Without pruning, 33 rules were tried; these included 20 separate rules plus repetitions due to renewed eligibility as new constraints were inferred. Of these, 11 were actually found applicable and were used to infer new constraints. When constraint targets were established and used, however only 8 rules were found eligible, and only 1 was used to infer a new constraint. Analysis without pruning was even more inefficient because cost estimates had to be found for additional alternative queries.

5.3 The stability of QUIST's control strategy

One measure of the effectiveness of a strategy to control inference is how well it constrains the search. In QUIST's case, the pertinent question is how many rules are tested during the analysis of a typical query. QUIST's control strategy has been effective for a relatively small set of approximately thirty-five rules. Would the strategy still be effective as the size of the database or the number of rules increases?

Our answer to the question is a qualified "yes". We shall argue that the number of rules that must be tested for a typical query is bounded, and that the bound hardly increases at all as the database or set of rules grow. We also offer some plausible arguments that the bound is small enough that the rules can be tested efficiently.

We make the following assumption about the rules in order to simplify the argument:

> A1. Simple rules. *Each rule is a production relating two attributes. That is, each rule is of the form* $C_1(A_1) \rightarrow C_2(A_2)$ *for two attributes* A_1 *and* A_2

In Section 4.4.2.1, we described how QUIST's rules are used during the analysis of a query. In this section, we follow that description in order to establish a suitable measure for the effort expended by QUIST on a typical problem.

First, we establish some terminology. A rule R is *associated* with an attribute A if and only if A is one of the two attributes constrained in the rule (there are just two, according to assumption A1). We designate by S(A) the set of rules associated with attribute A; this set need not be nonempty for every attribute.

Let us illustrate what can happen to a rule by means of an example. Suppose that attributes A_1 and A_2 are on constraint target relations, and that attribute A_3 is on a nontarget relation. Also suppose that the knowledge base contains the following rules:

R1: $(A_1 > 40) \rightarrow (A_2 > 200)$

R2: $(A_1 > 30) \rightarrow (A_2 > 50)$

R3: $(A_1 > 60) \rightarrow (A_2 > 300)$

R4: $(A_1 < 20) \rightarrow (A_2 > 150)$

R5: $(A_1 > 25) \rightarrow (A_3 > 100)$

Assume that at some point in processing, it is known that $(A_2 > 100)$ but that no constraint is known on A1 or A3.

Now suppose it is concluded (by rules other than R1 through R5) that $(A_1 > 50)$. All rules associated with A_1 (that is, those in $S(A_1)$), fall into one of five classifications illustrated by rules R1 through R5.

First, we exclude from possible use those rules that are not *promising* in the sense that they conclude a constraint about an attribute on a nontarget relation. Rule R5 is not promising because given a constraint on attribute A_1, it would conclude a constraint on attribute A_3 which is on a nontarget relation. By prior organization of the rules, it should be possible to determine unpromising rules once and for all at negligible cost.

Next, we see which rules are *applicable* in the sense that the rule's constraint on A_2 is implied by the newly inferred restriction on A_1. Rules R1 and R2 are applicable, but rules R3 and R4 are not. Furthermore, rule R4 can never be applicable, so it can be excluded from consideration at any subsequent point. By constrast, it is possible that rule R3 may become applicable if the constraint on A_1 is later strengthened as a result of additional inferences. We assume one "unit" of cost to check promising rules for applicability because of the work involved in comparing the rule constraints and the current restriction on the attribute. The test for potential future applicability falls out of the direct test for applicability so it costs no more.

Finally, we determine if the applicable rules are *effective* in the sense that they produce a new constraint. Rule R2 is not effective because the constraint it yields, ($A_2 > 50$), is weaker than the current restriction on A_2. Rule R1 is effective because it yields a new and stronger constraint, ($A_2 > 200$). In either case, another "unit" of cost is incurred comparing constraints on attribute A_2. In addition, both rules are now "used up" and excluded from further testing.

To generalize from this example, we assume:

- Unpromising rules incur no cost and are excluded from further testing.
- Inapplicable rules incur one unit of cost; only rules that are potentially applicable later are retained for further testing.
- Applicable rules, whether effective or not, incur two units of cost and are excluded from further testing.

Hence, the cost of analyzing a query can be determined as follows. For every attribute that is constrained once, either in the original query or by means of subsequent inference, the cost equals the number of inapplicable associated rules, plus twice the number of applicable rules. For every attribute that is constrained twice, the cost is the previous cost plus an additional cost figured only on the basis of potentially applicable rules left over after the first constraint was asserted. Costs for subsequent constraints are figured the same way, on the basis of a dwindling set of potentially applicable rules.

Therefore, the problem of determining the cost of analyzing a typical query becomes a problem of determining the following quantities:

1. the number of attributes that are constrained
2. the number of times each attribute is constrained

3. the number of associated rules that are promising

4. the number of promising rules that are applicable

5. the number of inapplicable rules that remain for subsequent testing.

As we stated above, we will not attempt to make an actual estimate of the cost of analyzing a typical query, but will argue that the cost is bounded, that the bound is stable with respect to the size of the database and the rule base, and that the bound is likely to be "reasonable". We make several more plausible assumptions:

A2. Simple queries. *Almost all queries constrain just a few attributes, no more than (say) five.*†

A3. Strong constraints. *Query constraints are often likely to be quite restrictive.*

A4. Nonuniform distribution of rules. *Some attributes have relatively many associated rules, many have relatively few or none.*

Based on these assumptions, we make one more crucial assumption:

A5. Limited inference. *Only a small number of attributes receive inferred constraints, and very few of these are constrained more than once.*

Our overall picture of inference in QUIST is as follows. A small number of attributes are restricted in the query (assumption A2). The process of generating constraint targets therefore does not yield a large set of targets. Consequently, only a relatively small percentage of the rules associated with the constrained attributes are promising, are tested, and incur a cost. The query constraints are probably strong, at least on the "important" attributes that are involved in many rules (assumptions A3 and A4). Therefore, very few new constraints are inferred and very few rules remain potentially applicable. Strong constraints probably lead to other strong constraints, so that there are few if any long chains of inference (assumption A5).

Returning therefore to the five quantities of interest described above, we are asserting that the number of contrained attributes is likely to be small and that each attribute is likely to be constrained no more than once. Concerning the quantities that involve numbers of rules, if the number of promising rules is reasonable, then the number of applicable and retested rules is reasonable too.

It is this last question of the number of promising rules that involves the growth of the database and the rule base. We make the following two assumptions about the effect of such growth:

---

†This seems to be the experience in systems like LADDER [Hendrix78].

A6. Growth of the database. *The growth in the number of items in the database has no effect on the number of rules.*

This assumption is actually rather obvious as the rules merely dictate permitted configurations of the data and are not otherwise linked to any aspect of the data, including the quantity of it.

As for whether the bound is reasonable or not, that depends on just how many rules are likely to be associated with the number of relations that are constraint targets in a typical query. There is no solid evidence from prior research to suggest what that number might be, but contemporary expert systems in artificial intelligence such as MYCIN [Shortliffe76] and PROSPECTOR [Duda78] have on the order of a few hundred rules for the entire system. This would certainly be a manageable number.

Chapter 6
The significance of semantic query optimization

In this final chapter, we discuss the significance of semantic query optimization in general and of its formulation in the QUIST system in particular. Our work advances specific ideas about the processing of database queries and about the organization of planning programs. It also serves as an important example of the fruitful interaction between research in artificial intelligence and research in databases. We also discuss the limitations of the research and make suggestions for future investigations.

6.1 Significance for database research

Semantic query optimization is significant for database research in tying together research on query optimization with research on the semantic integrity of databases. The synthesis provides a new and powerful method of query optimization. We discuss this in Section 6.1.1. In Section 6.1.2, we compare the work on QUIST with a related, more general proposal, called KBQP, of Zdonik and Hammer. We indicate that QUIST is a significant step forward because it provides specific answers about how semantic query optimization should be carried out and controlled in a context where query processing is relatively well understood, and because it has shown specifically by how much query processing can be improved using semantic reasoning.

6.1.1 The relationship of semantic integrity to query processing

The *semantic integrity* of a database is insured when the data in it are forced to meet *semantic integrity constraints* that reflect the real world application modelled by the database. The development of semantic integrity notions and the design of systems to enforce semantic integrity were sketched in Section 1.2.4. Through the work of Chang [Chang78], El-Masri [ElMasri80b], Hammer and McLeod [Hammer75], Roussopoulos [Roussopoulos77], and others, declarative formalisms have been applied to the purpose of stating general laws that express the semantics of a database.

The development of the ideas about semantic integrity constraints was motivated by one purpose,

that of making sure that the data in the database is meaningful. Our research advances another important and unforeseen use of these constraints. We have shown that:

> ***The semantic knowledge about a database expressed in semantic integrity constraints can sometimes be used to transform a database query into a semantically equivalent query that is much less expensive to process than the original query.***

This demonstration thus brings together the two apparently quite separate research areas of database integrity and query optimization.

The notion that general rules about the database can be applied during the processing of a query, and not just during the validation of updates, has appeared in other work, but not for the purpose of improving efficiency. For instance, in Chang's DEDUCE2 system [Chang78], general semantic rules are used to define *virtual relations* in terms of the basic relations that are stored in the database. When a DEDUCE2 query is processed, all virtual relations are transformed into the underlying basic relations. As with QUIST, DEDUCE2 checks whether the query poses conditions that violate semantic integrity constraints, but DEDUCE2 does not perform transformations for the sake of efficiency.

6.1.2 The organization and effects of semantic query optimization systems

The insight advanced by QUIST, that semantic integrity constraints can be used for efficiency transformations, has been introduced independently by Hammer and Zdonik [Hammer80] under the name *knowledge-based query processing* (KBQP). Their work resembles the QUIST work in three essential respects. First, of course, they propose that semantic knowledge about databases be applied to the problem of efficient query processing. Secondly, they suggest that the way to bring semantic knowledge to bear on this problem is by means of the transformation of queries into equivalent queries. Thirdly, they identify control of the query transformation process as crucial to the successful application of semantic knowledge to query processing.

However, QUIST makes important and original contributions in the introduction of the concept of semantic query optimization and in the organization and analysis of semantic query optimization systems. To identify these contributions, it is convenient to contrast QUIST with the KBQP proposal.

KBQP is intented to operate in the context of an abstract data management system that treats the database as a collection of sets of objects. The data model resembles the entity-relationship model [Chen76]. By contrast, QUIST operates with the relational model. The difference is significant because, as discussed in Chapter 2, research on the relational model has produced a body of query-processing expertise for which there is little counterpart in studies related to the entity-relationship model. Indeed, one of the significant demonstrations of our research is that:

Factors that govern the cost of processing a relational database query can be expressed as expert rules that can help control the query transformation process of a semantic query optimization system.

It is worth noting that this result is the product of taking an artificial intelligence "perspective" toward results in database research.

As noted above, the KBQP proposal recognizes that in order to maintain the overall efficiency of the system, it is necessary to perform only those query transformations that may lead to reduced query processing costs. Hammer and Zdonik postulate a set of what they term cost-reducing *techniques* to control transformations. For example, the aim of one such technique, called *domain refinement*, is to convert the domain of a restriction expression into a smaller one whose members are more readily accessible. The technique of domain refinement seems intuitively plausible. Indeed, it corresponds in the relational context to QUIST's *index introduction* transformation (Section 4.4.1.3). What the research on QUIST has done is to go beyond intuitive plausibility in the context of a well-developed model of query processing (Section 5.1), leading to the assertion that:

Several classes of transformations of relational database queries that reduce the cost of processing have been identified, and the reduction they produce in the cost of processing has been estimated quantitatively based upon well-developed models of query processing.

To control the application of their cost-reducing techniques, Hammer and Zdonik propose a multiprocesssing control structure. At the start of analyzing a query, a separate process is set up for each technique applied to each subexpression in the original query. Each process is assigned a priority based upon heuristics that reflect the presumed likelihood that the particular technique will succeed and produce an improvement in the particular subexpression. An example of such a heuristic is: "assign a low priority to a process that involves domain refinement applied to an expression that does not appear in any statements about subset relationships in the knowledge base". Hammer and Zdonik acknowledge that the number of processes is apt to grow large. The reason they propose such an elaborate control structure in spite of this is their belief that it is necessary to reason about transformation goals at every step in the analysis of the query.

QUIST controls the transformation process quite differently (Section 4.3). It forms constraint targets in a separate analysis before it attempts to infer any constraints. Because of this separation, the inferences that produce transformed queries are carried out in a data-directed rather than a goal-directed manner. That is, QUIST reasons forward from known constraints without having a precise goal for that reasoning. This is not to say, however, that QUIST does not identify which constraints would be desirable. This is exactly what QUIST does when it identifies constraint targets (its so-called *planning* step, Section 4.3.1). Rather, QUIST uses goal information to cut off unpromising lines of inference.

The result is that QUIST's control strategy is much less elaborate than the one proposed for

KBQP. In Section 5.2, we report on experiments that show QUIST's control strategy to be effective, at least for a limited sample of QUIST's class of relational database queries. In Section 5.3, we argue further that the control strategy remains effective under reasonable assumptions about the complexity of the semantic rules and the growth of the database.

The KBQP approach to control reflects the philosophy that determining a suitable query transformation is a very complex problem and that the possible improvement in the query warrants an elaborate and possibly expensive analysis. QUIST's approach reflects a different philosophy: keep the analysis simple at the cost of missing some desirable transformations. At first glance, KBQP's approach seems more general. Yet, QUIST's approach seems appropriate where, as in the case of its particular class of relational database queries, where storage and access conventions are well established and cost factors are well understood. The point is that QUIST can make reasonable assumptions about the frequency and the consequent importance of certain kinds of constraints (namely, those on single attributes, particularly indexed attributes). Its knowledge base and its control strategy are based on these assumptions. It may be that as other classes of queries and other means of storing and accessing data are better understood, new QUIST-style heuristics can be developed and QUIST's approach will prove effective. Which philosophy is more appropriate for semantic query optimization in general can only be determined by further research. However, it can be said that:

> ***There is evidence that a simple control strategy that uses forward reasoning limited by a set of previously computed constraint targets is effective for semantic query optimization in attribute/constraint relational queries.***

KBQP is a design proposal that would probably require new machine architectures for cost-effective implementation. By contrast, QUIST has been implemented and tested on a range of queries. It builds explicitly on assumptions and models of contemporary research in query optimization for relational databases as implemented on current generation serial architectures.

We can summarize the relationship of semantic query optimization to the methods we have called *conventional* query optimization as follows:

> ***Semantic query optimization makes it possible to achieve substantial improvements in the efficiency of processing that are not achievable by conventional techniques. At the same time, though, semantic query optimization can be viewed as extending the usefulness of conventional methods in the sense that the purpose of producing semantically equivalent queries is to create new opportunities to apply conventional query optimization techniques.***

Finally, we should note:

> ***The development of semantic query optimization demonstrates the fruitfulness of***

> ***investigating certain database problems from the point of view of artificial intelligence research.***

The development of semantic query optimization is part of a growing awareness of this point [Brodie81] that is highly significant for database research.

6.2 Significance for artificial intelligence research

As formulated in the QUIST system, semantic query optimization is significant in two major respects. First, it suggests a new *problem reformulation* approach to the task of producing a "good" plan when there already is an existing planning program to produce correct plans. Second, it provides an example of *intelligent database mediation* by providing intelligent assistance in the best use of database resources. In Section 6.2.1, we identify a conventional query optimizer as a planning program. We discuss recent research in planning and problem solving in which the issues of efficiency and explicit control of problem solving emerge. Finally, we contrast QUIST's approach on these issues with the approach taken in other planning systems. In Section 6.2.2, we define the database mediation task and note how QUIST has supplied one part of the desired function.

6.2.1 The reformulation of problems for better solutions

Given a database query stated in logical terms, the problem of query optimization is to specify an efficient way to process that query in the physical database. That is to say, the problem of query optimization is exactly what is referred to in artificial intelligence research as *problem-solving*. Problem-solving is the determination of a sequence of actions to satisfy a goal. In query optimization, the goal is to obtain some data or to check the truth of some assertion. The actions through which the goal can be satisfied are operations in the physical database such as segment scans and indexed scans (Section 2.3).

The resemblance between a query optimizer and an artificial intelligence problem-solving program is illustrated by the System R query optimizer. As noted in Section 2.4, System R's optimizer analyzes the processing of an n-relation query as a sequence of processing 2-relation queries. Thus, each 2-relation query can be regarded as an *abstract step* in the plan to perform the desired retrieval. One of the main tasks of the optimizer is to pick the best way to carry out each 2-relation query. There may be many ways to do this; in fact, [Yao79] describes a model that can generate 339 different methods to carry out a 2-relation query. Thus, the optimizer must *refine* the abstract step in the best available way. The refinement of abstract plan steps is a fundamental part of all recent planning programs whether they are based on *hierarchical planning* (NOAH [Sacerdoti77]), *best-first search* (LIBRA [Kant79]), or *orthogonal planning* (MOLGEN [Stefik80]).

The other task of the System R optimizer is to choose the best sequence in which to perform the 2-

relation queries. Sequencing and refinement are related tasks. For example, some processing steps compute a result in an order that differs from the original order of the data from a given relation. Any index on that relation is no longer usable for later steps. Hence, certain refinement choices are lost. Conversely, a single refinement choice for a particular step may so dominate the alternatives as to force the step to be performed early in the sequence in order to avoid possible invalidation. This interaction between between sequencing and refinement is also seen in programs like LIBRA [Kant79].

In the QUIST system, it is assumed that a conventional query optimizer is available to carry out the refinement of a logically stated query into a plan for execution in the physical database. What is significant about QUIST is the following:

> ***A semantic reasoner can be applied as a preplanner that can result in the production of better plans by its associated planning program, without complicating the planning program itself.***

To understand the significance of this for planning problems in general, let us review some recent planning programs in more detail. The review focusses on two issues: the role of efficiency considerations in planning, and the control of the planning process itself.

The PEGASUS program of Sproull [Sproull77] was one of the first planning programs to address efficiency issues directly ([Garvey76] provides another example). Sproull's chief concern was to integrate the symbolic planning methods of artificial intelligence with the considerations of *utility* developed in decision theory. The basic approach taken by PEGASUS was to conduct a search of plan alternatives using a utility function to measure the promise of partially completed plans. The utility function did more than this, however. It also provided the basis for judging the relative value of further planning, of obtaining more information about the (uncertain) environment, and of carrying out proposed plan steps. Thus, the PEGASUS planner controlled its own activities using the same utility functions it employed to select the best plan. The overall goal of PEGASUS was to achieve optimal behavior measured in terms of the *combined* utility of the execution of the completed plan and of the planning process itself.

Kant's LIBRA program [Kant79] also considered efficiency explicitly. Its goal was to take a high-level description of a program and to transform it into an efficient program that could actually be executed. Knowledge about how to transform a program was contained in *coding rules* developed by Barstow [Barstow79]. LIBRA's task was to decide which of possibly many coding rules to apply at any point. It used *efficiency rules* to do this (and in this respect is very much like QUIST). The efficiency rules reflected both heuristic and analytical estimates of the cost of alternative refinements. In addition, LIBRA used *resource allocation rules* to decide which part of the program description to refine first. Choosing to refine some parts before others could greatly reduce the number of refinement alternatives that had to be considered.

The MOLGEN program of Stefik [Stefik80] advanced the notions of *metaplanning* and *constraint*

posting. The idea of metaplanning is that the planning process itself should be controlled by a similar planning process (a similar notion appears in [Hayes-Roth78]), responsible for such activities as focus of attention at the planning level. Constraint posting is the idea that decisions should be postponed until the constraints arising from commitments or guesses elsewhere in the developing plan are propagated. This reduces the number of alternatives that must be considered.

These programs illustrate several major themes of current research in problem solving and planning:

- Planning proceeds by adding constraints to a partially completed plan.
- The programs reason explicitly about the control of the planning process.
- Decisions about how to refine a particular segment of a plan are intermixed with decisions about what the planning program should do next.
- In many cases (PEGASUS and LIBRA, for example) it is desired to produce not just a correct plan but a "good" plan, and furthermore, it is desired to that the planner itself be efficient.

In query optimization, including semantic query optimization, we are obviously concerned with the quality of the final plan, as measured by its efficiency. We are also concerned with the efficiency of the planning process. Where QUIST differs from contemporary planning programs is in its approach to finding an efficient plan. Rather than integrating decisions about the planner's focus of attention with decisions about the choice of refinement, including those choices that bear on efficiency, QUIST moves considerations of efficiency into a preplanning step. In this step, constraints are added to the statement of the problem itself. The constraints are added not as the result of elaboration of a plan step, but rather for the express purpose of having the planner work on a new but equivalent problem for which a more efficient plan may be generated. In other words:

> ***A preliminary reformulation of a problem statement can be used to achieve a more efficient solution to the problem, thereby avoiding explicit and possibly costly analysis of efficiency factors during the actual process of producing the solution.***

The result is that the conventional query optimizer, viewed as a planner, can be much simpler than it would have to be if it tried to add new constraints to the plan in order to make the plan more efficient.

Is it really necessary to simplify the planner in this manner? Both Sproull and Kant have claimed that their systems not only produce efficient plans but do so with an efficient planner. In fact, despite some investigation of the issue, the cost of planning is not a crucial factor in PEGASUS's travel planning domain nor in LIBRA's program synthesis domain. That is not to say that an integrated control strategy may not be appropriate. It does suggest that further investigation is needed to determine where that strategy is worthwhile. In any event, both PEGASUS and LIBRA work with

essentially fixed problem statements; new constraints enter only in the refinement to executable plans. (Interestingly, the planner reported in [Hayes-Roth78] does not have a fixed problem statement; the planner is free to choose which of many tasks to perform. The "goodness" of the plan is loosely related to how many tasks can be carried out using the completed plan.)

The separation between problem reformulation and problem solving raises an issue that is not present in typical planning programs: how is problem reformulation controlled? In QUIST, problem reformulation (the semantic transformation of the query) is controlled by means of the constraint target list (Section 4.3.1). The constraint targets are determined from knowledge about the possible opportunities for finding less expensive ways to search the files involved in the query. In the terminology of planning:

> ***The process of reformulation of the problem for the sake of efficiency can be guided by knowledge about the cost of processing alternative refinements of abtract plan steps.***

That is, there is a two way flow of information. Not only does problem reformulation change the class of possible plans to include more efficient plans, but also the information about the cost of plan operators that the planner uses can be abstracted to guide the reformulation process.

To summarize, then, semantic query optimization as formulated in the QUIST system offers a new method for achieving a "good" solution to a problem when a method for finding correct solutions already exists. The new method consists of reformulating the statement of the problem into an equivalent form for which better solutions may exist. The process of reformulating the problem statement is controlled by using an approximate model of the kinds of solutions produced by the associated problem solver.

6.2.2 Intelligent database mediation

A user who wishes to access a database in order to solve a problem faces several difficulties. For one thing, the user may not know what information is contained in the database. For another, he may not know what concepts the database uses in general and what terminology is used to refer to them. Even if the user understands the database's structure and terminology, he may not know how they relate to his own concepts and terms for the problem domain.

In conventional database installations, the user must either puzzle out these problems on his own, or else he has recourse to the services of a database analyst or liaison. The analyst *mediates* between the database resources and the user solving a problem. The analyst applies knowledge of both the problem domain and of the capabilities and limitations of the database to pose the most effective and easily processed queries that can help solve the original problem. The analyst supplies certain knowledge about the database which the user lacks in order to make the most effective use of the database. Of course, the analyst must know enough about the problem domain in order to do this sensibly.

With the development of interactive query systems, it is expected that average users will interact more directly with databases, without the aid of a database analyst. It is clear, however, that better facilities must be created to perform some of these *intelligent database mediation* functions automatically.

Some aspects of intelligent database mediation have been explored. McLeod's IFAP (Interaction Formulation Advisor Prototype) [McLeod78] supplies knowledge of the classes of entities known to the database about which a user can pose queries. In the LADDER system [Hendrix78], a user's natural language query is transformed into a retrieval language query to the appropriate network site and database. That is, one of LADDER's functions is to supply knowledge of the distribution of data among sites and databases, knowledge that the user lacks.

Semantic query optimization is significant as an *application* of artificial intelligence methods to one aspect of the intelligent database mediation problem:

> ***As an intelligent database mediator, a semantic query optimization system employs detailed knowledge of semantic constraints on the data and detailed knowledge of the physical organization of the database, knowledge that a user should not be expected to know or to be able to use.***

In addition,

> ***Semantic query optimization is the first effort to apply semantic reasoning to the task of providing efficient access to pre-existing computer resources.***

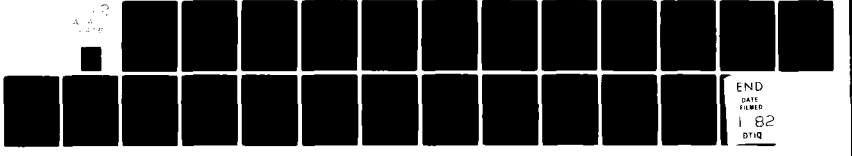
We are not claiming that the present research has discovered the problem of intelligent database mediation nor that it has devised entirely new solutions to that problem. Rather, the present research should serve to encourage additional applications of artificial intelligence techniques to database mediation and other database problems.

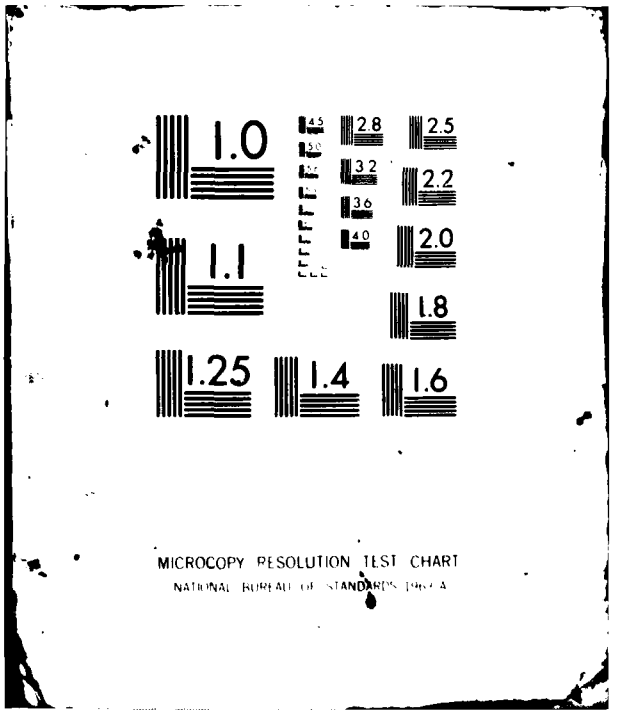
In the future, we will want computer systems to be increasingly knowledgeable not just about the answers to specific questions, but also about the range of knowledge sources which it can access and the ways in which those sources can be used. The research reported here is a step in that direction.

6.3 Limitations and directions for future research

Database retrieval is a very important activity. Semantic query optimization holds the promise of substantial improvements in this activity. Therefore, it is worthwhile examining how the ideas advanced in this research are limited, and how their future usefulness might be extended. We are particularly concerned with how semantic query optimization can be extended to other data models and system architectures, how additional kinds of semantic knowledge can be employed for efficient

AD-A108 735 STANFORD UNIV CA DEPT OF COMPUTER SCIENCE F/G 6/4
QUERY OPTIMIZATION BY SEMANTIC REASONING.(U)
MAY 81 J J KING N00039-80-G-0132
UNCLASSIFIED STAN-CS-81-857 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

processing of queries, and how methods to control semantic query optimization may have to be extended.

6.3.1 Data models and database architectures

The QUIST system operates in the context of a subclass of the relational model of data which we have referred to as an *attribute/constraint* model (Section 4.2.1). We have indicated that this includes an important class of queries. Yet, it may be desirable to extend the system so that it is *relationally complete* [Codd71]. To do this, it would be necessary to drop the assumption that there is a single logical path between any two relations. This would require a somewhat more complicated representation of semantic rules, because the logical path between relations would have to be specified. Whether the extra complexity would be merited by the frequency of queries outside the range now covered by QUIST would have to be studied. The difficulty would be to retain the potential improvements of semantic transformations without adding the complexity of a general-purpose theorem prover.

Our research centered on the relational model because that model has been the focus of attention of much recent research on query processing. However, semantic reasoning can certainly be applied in the context of other data models. For example, the principle of "pushing a constraint up a hierarchy" (Section 4.4.1.3) certainly makes sense in a hierarchical or network database.

We also adopted a conventional model of data storage and access (Section 2.3). This model or models like it is the basis for most research in query optimization. However, there is growing interest in query optimization in distributed databases ([Epstein78]) and in unconventional database machines ([Shaw80]). More generally, there is recent research ([Lenat79], [Katz80]) that extends the ideas of Wiederhold ([Wiederhold77]) on the notion of the *binding* of semantic knowledge to data structures. The aim of this research is to develop abstract descriptors and rules with which to reason about the ease or difficulty of realizing the physical counterpart to a logical expression. If this effort is successful, it would be an appropriate vehicle to generalize the heuristics of QUIST to apply to multiple databases and unconventional architectures.

6.3.2 Semantic knowledge

The semantic knowledge used by QUIST involves constraints on particular *values* stored in the database. However, there are other kinds of constraints that could be used for semantic query optimization.

Cardinality constraints specify the minimum or maximum number of individuals in some entity class that can be associated with an individual in some other entity class by means of a particular type of relationship. An example is: "every freshman and sophomore must have at least two faculty advisors." *Dependence* constraints can be viewed as cardinality constraints in which at least one

related entity must exist. Dependence and cardinality constraints are particularly significant in terms of the *structural integrity* of databases [ElMasri80b]. For example, the constraint that "every manager must manage exactly one department" regulates updates and deletions of manager and department entities; the deletion of a department entity forces the deletion of related manager entities, and no manager entity can be inserted for which the designated related department entity does not exist in the database. However, the constraint does not determine which managers can be related to which departments. Among the structural constraints are the widely discussed "functional dependencies" and "multivalued dependencies" [Ullman80].

To see how cardinality and dependence constraints could be used for semantic query optimization, consider the constraint "every student except those with an independent study major has at most two advisors." Suppose the query is: "what are the name and faculty rank of the advisors of history majors?" One way to process the question would be to find each history major in turn, and then to find each of his advisors and print his name and rank. However, we know that there are at most two advisors for each student, so the search for a given student's advisors can stop when the second advisor has been found. Notice that the ability to exploit the constraint on the number of advisors requires a different control strategy than QUIST's. Specifically, it requires more direct control of the query processor itself so that, for instance, a limit on the number of hits from some file can be set and reset as needed. By contrast, QUIST works entirely at the level of transforming the "surface level" of queries. The only thing that the semantic optimization component passes down to the query processor is a query, and not any instructions on how to process it.

Another kind of semantic knowledge is what can be termed *approximate* knowledge, knowledge that is probabilistic or about which there is some uncertainty. It includes the heuristics or rules of thumb that help experts to reason effectively in their area of expertise. Approximate knowledge could be applied to semantic query optimization by using a somewhat different strategy than QUIST uses, one that is itself heuristic in nature. Suppose it is known that most supertankers are registered in Panama, Liberia, or Greece, and suppose a query asks for the names of three supertankers carrying crude oil to Italy. In that case, it is likely that the names of three qualifying supertankers can be found merely by examining those registered in Panama, Liberia, or Greece. If the registration information is well supported in the database (say, by an index) and if there are indeed three supertankers registered in one of those countries and that are currently carrying crude oil to Italy, then it is proper and effective to transform the question so that it references the country of registration.

This strategy offers no guarantee that the substituted query gives the same answer as the original query. Therefore, a more sophisticated system is needed to apply this new strategy effectively. For instance, suppose there are 100 supertankers. If we know that "most" supertankers are Liberian, then it seems likely that questions that request 5 supertankers can be answered merely by referencing Liberian tankers. However, it may not be effective to process questions that request 95 supertankers by looking first only at Liberian tankers. If the required number of supertankers are not found among the Liberian ones, the search must be renewed among all supertankers. The system must be

sophisticated enough to determine when it is probably worthwhile adding constraints to the query based on approximate knowledge.

Finally, we consider the use of knowledge not about the semantics of the domain but about the relationship between concepts defined within the database itself (the familiar notion of *logical views*). For instance, suppose that in a university database the predicate INSTRUCT is a derived relationship between professors and students defined in terms of fundamental predicates TEACH and ENROLLED-IN as follows:

$$\forall p,s\ \text{INSTRUCT}(p,s) \equiv \exists c.(\text{TEACH}(p,c) \wedge \text{ENROLLED-IN}(s,c))$$

This says that a professor p instructs a student s if and only if there is some class c that the professor teaches and in which the student is enrolled. Let us now consider the query:

"What professors instruct all the students whom they advise?"

which we render as:

$$\{p \mid \forall s.\ \text{ADVISE}(p,s) \rightarrow \text{INSTRUCTS}(p,s)\}$$

The strategy for this query would be to eliminate all professors who advise more students than they instruct, for in that case, they certainly can't instruct every student whom they advise. We can conservatively assume that each student is enrolled in no more than one course taught by any professor. Then for every professor who satisfies the query, it must also be true, from the definition of "instructs", that he instructs fewer students than the product of the maximum number of students in any one class and the maximum number of classes taught by any professor. Let Cnt(S) stand for the number of items in set S, and Max(x,F(x)) stand for the upper bound on function F(x) for any value of x. In addition, let I(p) be the total number of students instructed by a professor, p. That is:

$$\forall p\ I(p) \equiv \text{Cnt}(\{s \mid \text{INSTRUCT}(p,s)\})$$

Then the conservative upper bound can be expressed as:

$$\forall p\ I(p) \leq \text{Max}(c,(\text{Cnt}(\{s \mid \text{ENROLLED-IN}(s,c)\}))) * \text{Max}(p,(\text{COUNT}(\{c \mid \text{TEACH}(p,c)\})))$$

If, for instance, there is a maximum enrollment of 20 students in any course, and a maximum teaching load of 3 courses per professor, then we can eliminate from consideration any professor who advises more than 60 students. This conservative bound can be tightened as more information is gathered during query processing. Thus, if professor X teaches 2 courses, one with 12 students and one with 18, he can be eliminated if he advises more than 30 students, rather than the conservative bound of 60. As this strategy uses cardinality constraints, it relies on more detailed control of query processing than QUIST does.

6.3.3 Control of semantic query optimization

In Sections 6.1.2 and 6.2.1, we argued the merits of organizing a semantic query optimization system as a preplanner. That is, we advocated performing all semantic transformations of the query prior to generating the sequence of steps for actually carrying out the query in the physical database.

However, we saw an example in the last section in which knowledge about the distribution of current data values could provide cost-reducing constraints. In this section, as in Section 6.2.1, we recommend that further research be conducted on the question of under what conditions control of semantic query optimization should be integrated with the processing of the query itself. As further justification for such research, we offer two more examples of data-dependent semantic query optimization.

Consider, for example, a database that contains data about ships and their movements. Assume that the most frequent queries concern the current status of American ships, so that a small file containing duplicate information about their most important current voyage attributes is maintained. Whenever a position report is received on an American ship, both the regular file and the duplicate "highlights" file are updated. Suppose that a user poses the query:

"Where is the fastest tanker?"

If the nationality of the ship is stored with its speed and shiptype, then we can check whether the ship is American. If it is, then we only have to look for its position in the small file of American ships. Otherwise, we have to look through the larger file of position reports for all ships. If the fastest tanker happens to be American, then in effect the original query can be transformed into:

"Where is the fastest American tanker?"

But this transformation is only supported in the current state of the database. There is no integrity rule prohibiting the insertion of another record representing a faster tanker of another nation. Thus, this transformation is inherently dependent upon the current state of the database.

The preceding example and the one in the last section do not actually use any rules about the application domain; they only use relationships internal to the database. Yet, the current contents of the database can affect the application of a domain rule as well, as the following example illustrates. Assume we have simple relational database:

SHIPS: (Shipname Shiptype Length Draft Capacity)

PORTS: (Portname Country Depth Facilitytype)

VISITS: (Ship Port Date Cargo Quantity)

and the following two semantic integrity rules based on domain knowledge:

Rule R1. "A ship can visit a port only if the ship's draft is less than the channel depth of the port."

Rule R2. "Only liquefied natural gas (LNG) is delivered to ports that are specialized LNG terminals."

Assume that each relation is implemented as a single file on its own data pages. The VISITS file has a clustered index on Cargo. Now consider a query that requests the ships, dates, cargoes and quantities of visits to the port of Zamboanga. According to our semantic query optimization heuristics, it is desirable to infer a constraint on Cargo from the given constraint on Port.

Imagine that instead of performing semantic transformations in a preplanning phase, there is integrated control of semantic transformation and data retrieval. Control of the process of inferring cost-reducing constraints can then be viewed as control of the moves in a space of constraints on attributes. Constraints can be moved either by applying a rule, by retrieving items restricted on one attribute and observing their values on other attributes, or by matching constraints on attributes defined on the same underlying set of entities.

Continuing the example, starting with a constraint on the Port attribute of VISITS, new constraints can be found by retrieving from VISITS or by assigning the value "Zamboanga" to the Portname field of PORTS. The first choice is rejected because the objective is to reduce the cost of that very retrieval. With a constraint on Portname in PORTS, a retrieval from PORTS can be performed. In this case, just a single record will be obtained because Portname is the unique identifier in that file. With appropriate access methods, such as hashing, the retrieval will be very inexpensive.

When the PORTS record for "Zamboanga" has been obtained, rules R1 and R2 may apply. If rule R2 applies, that is, if Zamboanga is a specialized liquefied natural gas terminal, then a strong constraint will be obtained on the goal attribute Cargo, and retrieval from VISITS will take place by means of an indexed scan rather than by means of a more expensive sequential scan. If the data on Zamboanga does not support that inference, then other inference paths beginning with rule R1 will have to be considered. This illustrates the possible dependence of retrieval planning on the current contents of the database.

Whether or not such elaborate control is worthwhile is certainly open to question. It depends in part upon what kinds of processing options are available; it seems more likely, for instance, that an integrated strategy makes more sense in a distributed database with redundant files. The point of these examples is simply to indicate the value of further research on this issue.

6.4 Conclusion

This research has introduced a new method to reduce significantly the cost of processing database queries. The method uses semantic knowledge that is otherwise used to insure the validity of database entries. It applies techniques of artificial intelligence to the problem. At the same time, it suggests a new approach to problem solving when the quality of the desired plan is important and there already exists a generator of correct plans. This approach is to reformulate the problem statement as an equivalent problem which may have a better solution.

To be useful in future database systems, the work presented here must be extended to additional models of physical data storage and access and to a wider range of logical data models. Also, experience is needed with actual database systems to test further the promising results obtained under laboratory conditions; tests of query processing methods are generally run on small sets of invented examples, but this is not a suitable practice for future work. Additional research is needed to investigate when a problem reformulation strategy can be applied to the task of finding good solutions to problems.

Whatever the particular merits or shortcomings of semantic query optimization and the QUIST system, the research presented here suggests the value of work at the intersection of database management and artificial intelligence. These fields are important and exciting and have a great deal to offer to each other.

Appendix A
The QUIST query language

A.1 Syntax of the QUIST query language

In the following description, the metasymbol "+" means one or more instances of the type so designated, and the metasymbol "|" means a choice between the items it separates. In the actual QUIST language, the tokens ONEOF and NOTONEOF are used instead of the symbols $\in$ and $\notin$, respectively, that are used in the examples throughout this report.

<query> ::= (<selections> <restrictions>)

<selections> ::= (<attribute>+)

<restrictions> ::= (<restriction>+)

<restriction> ::= (<attribute> <constraint>)

<constraint> ::= <string constraint> | <integer constraint>

<string constraint> ::= (ONEOF (<string>+)) | (NOTONEOF (<string+>))

<integer constraint> ::= (<interval>+)

<interval> ::= (([GT|GE] <integer>) ([LT|LE] <integer>)) | ((<comparator> <integer>))

<comparator> ::= GT | GE | LT | LE | EQ | NE

A.2 Semantic restrictions on the language

1. An attribute can appear only once among the selections or among the restrictions.

2. An integer-valued attribute can only be constrained by an integer constraint, and a string-valued attribute only by a string constraint.

3. The intervals of an integer constraint must not conflict. This requirement is enforced as follows. If the constraint is ((Comp1 Value1) ... (CompN ValueN)), then Value1 < Value2

< ... < ValueN. Furthermore, each GT/GE term that is not last in the list must be followed by a LT/LE term (possibly with some intervening NE terms), and each LT/LE term that is not last in the list must be followed by a GT/GE term (possibly with some intervening EQ terms).

Appendix B
QUIST and the relational calculus

In this appendix, we show that the queries, semantic equivalence transformations, and integrity rules of QUIST are special cases of relational calculus queries, semantic equivalence transformations, and integrity rules, respectively. To do this, we indicate how to determine the relational calculus query that corresponds to a QUIST query. We also show how the process of semantic equivalence transformation in QUIST is a special case of the process for relational queries.

B.1 Generation of a relational calculus query from a QUIST query

A QUIST query consists of a qualification, which is conjunction of constraints on database attributes, and an output list, which is a list of attributes whose values are requested. Hence, a QUIST query corresponds to an *open query* of the relational calculus, as defined in Chapter 3. A QUIST query is simpler than a relational query chiefly in two respects: join terms need not be stated explicitly, and only conjunctions of attribute constraints can be expressed. The major difference in form arises from the assumption in QUIST that every attribute is associated with a single virtual relation, hence no relation need be specified. Relational calculus queries, on the other hand, employ tuple variables that range over explicitly specified database relations.

To generate the relational calculus query that corresponds to a QUIST query, it is therefore necessary to determine what real relations are involved in the query and what join terms are needed to link them together properly. Because only conjunctive queries are permitted, every relation in the database that is involved in the query plays one of only four possible roles:

1. some of its attributes are constrained but none are in the output list;
2. some attributes are in the output list but none are constrained;
3. some attributes are constrained and some are in the output list;
4. its attributes are neither constrained nor designated for output, but it is joined between two (or more) other relations.

A tuple variable must be generated for every relation that is involved in the query. If some attributes of the relation are designated for output (Cases 2 and 3), then the variable appears in the

relational query target list. If not (Cases 1 and 4), then the variable appears as an existentially bound variable within the qualification.

If some of the relation's attributes are constrained (Cases 1 and 3), then it is necessary to generate a restriction term in the corresponding tuple variable. The restriction term is the conjunction of the restrictions on the attributes of the relation.

A QUIST query does not mention any relations that are joined to others but are not otherwise involved in the query (Case 4). The fact that these relations are involved is determined in the course of generating the required mappings (join terms) among the other involved relations (Cases 1, 2, and 3).

It is fairly simple to generate the mappings among the query relations because there is one and only one logical access path or mapping between any two relations in the database; this is a distinguishing characteristic of QUIST and of other attribute/constraint relational query languages. Consequently, we can choose any relation involved in a query and regard it as the root of a tree of relations. Every other relation in the query can be reached via some unique path. Indeed, the structure of a query should be viewed as a subtree of QUIST's virtual relation. The virtual relation, defined in Chapter 4, is a tree structure in which all the real database relations are linked by way of uniquely specified sequences of joins.

B.2 The generation algorithm

We use the tree-structure property of the virtual relation to generate the relational query from the QUIST query. The major steps of the algorithm are:

- Step 1. Determine directly from the QUIST query which database relations have either constrained or output attributes (Cases 1, 2, and 3).
- Step 2. Designate one of these relations as the root of the tree of query relations.
- Step 3. Choose a relation found in Step 1. Generate a tuple variable for it and link it up to the relations that have already been chosen by generating the appropriate join terms. Generate a restriction term for the relation if necessary. Repeat until all such relations have been chosen.
- Step 4. Formulate the relational query from the tuple variables, restriction terms, and join terms.

We now describe the steps of the algorithm in more detail.

The first step finds all relations that are constrained or part of the output. Go through restrictions in the QUIST qualification. If the attribute in the current restriction is associated with a relation X that has not yet been encountered, then add X to the set S_c of constrained relations. Next, go through the QUIST output list. For each new relation X encountered, add X to the set S_{out} of output relations. Finally, let S_{all} be $S_c \cup S_{out}$, the set of all relations so far involved in the query.

In the second step, we choose any relation X from S_{all} to be the root relation to which all the other relations will be linked. We generate a tuple variable x for this relation. If X is in S_{out}, the set of relations involved in the output, then variable x is placed in set Q_t, the set of variables that will be in the target list; the subscript "free" conveys the idea that variables in the target list appear free in the qualification of the relational query. Otherwise, x is placed in $Q_{\exists}$, the set of variables that will be existentially bound in the qualification. If X is in S_c, the set of relations involved in QUIST constraints, then a restriction term P(x) is generated. P(x) is the conjunction of restrictions on attributes associated with X. For example, if the restrictions ($A_1 \in$ {"a" "b"}) and ($A_2 \in$ ([50 100)) are the QUIST constraints associated with relation X, then P(x) is given by:

$$P(x) \equiv (x.A_1 = \text{"a"}) \vee (x.A_1 = \text{"b"})) \wedge ((x.A_2 \geq 50) \wedge (x.A_2 < 100)).$$

The third step is the most complicated and requires further discussion and some terminology. The key concept is that of the logical access path or *mapping* between any two relations X and Y. A QUIST mapping is the unique expression of join terms by which the two relations can be linked. Assume that tuple variable x ranges over relation X and tuple variable y ranges over relation Y. Let M(x,y) (= M(y,x)) denote the mapping between X and Y in terms of these tuple variables. If X and Y are "neighbors" in the sense that they are permitted to be joined together directly, then M(x,y) is simply

$$M(x,y) \equiv J(x,y)$$

where J(x,y) is the join term ((x.A) = (y.B)) for some prespecified attributes A and B of X and Y, respectively. This specification is part of the definition of the QUIST virtual relation.

Suppose, however, that the virtual relation is defined in such a way that X and Y are only allowed to be linked via intermediate relation Y_1. This means that the relational counterpart of the QUIST query involves relation Y_1 as a necessary joining "bridge" between X and Y. The relational query now involves a term that represents this mapping between X and Y:

$$M(x,y) \equiv \exists y_1 \mid J(x,y_1) \wedge J(y_1,y).$$

That is, for every qualifying pair of tuples (x,y) from X and Y there must be some tuple y_1 in Y_1 that supports the mapping by way of the two prespecified joins.

In general, some sequence of relations $Y_1, Y_2, \ldots, Y_n$ intervenes between X and Y in the predefined mappings of the virtual relation, where we adopt the convention that the lower the subscript, the closer the relation is to X. The mapping expression is then given by:

$$M(x,y) \equiv \exists y_1 \ldots \exists y_n \mid J(x,y_1) \wedge J(y_1,y_2) \wedge \ldots \wedge J(y_n,y)$$

where the intermediate conjuncts correspond to the prespecified allowable joins.

Even though relations X and Y are involved in the constraints or output designated by the QUIST query, it may be that some or all the relations Y_i that connect X and Y are not involved in that way. However, these relations must be specified in the relational query counterpart to the QUIST query. They constitute the Case 4 relations defined above.

The idea of this next step of the algorithm is to link up the relations in S_{all} one at a time by linking each one to the root relation X, introducing new "bridge" relations as needed. Every time a new relation is selected for linking or is introduced as a bridge relation, a new tuple variable is generated.

Some complications may arise in the linking process. For one thing, some or all of the necessary mapping expression may have been generated when a previously chosen relation was linked up. Intuitively, this occurs when the new relation lies farther out along the same branch from X as the preceding relation, or when the new and preceding relations have a common ancestor relation between them and X. Therefore, each linking step only introduces as much of the mapping expression as necessary. One other possible complication is that the new relation may have been introduced already as part of the bridge between X and a preceding relation. In this case, a new tuple variable should not be generated for it.

We now resume the description of the third step in the algorithm. *Let Y be the new restriction in* S_{all} that is to be linked up. Assume for the moment that Y has not been previously introduced as a bridge relation. Therefore, we generate a new tuple variable y to range over Y. If Y is a Case 1 relation, constrained but not part of the output, then variable y is placed in the set $Q_{\exists}$, as it will appear existentially bound in the relational query. Otherwise, y is placed in Q_t because it will be in the query's target list.

We must now introduce some part of the mapping expression M(x,y). If we denote X by Y_0, then the definition of the virtual relation specifies that there are n relations Y_0 through Y_n, $n \geq 0$, between X and Y. Let Y_k, $0 \leq k \leq n$, be the relation with the highest subscript among these relations that have already been linked into the query; that is, Y_0 through Y_k have already been linked in. Therefore, it is only necessary to complete the link from Y_k to Y.

In other words, instead of generating the mapping expression M(x,y), we generate the mapping expression $M(y_k,y)$. In addition, we generate tuple variables y_{k+1} through y_n, and place these in the set $Q_{\exists}$ because they enter the relational query as existentially bound variables. Of course if $n = 0$, meaning that X can be joined directly to Y, or if $k = n$, meaning that all intervening relations have already been linked in, then no new tuple variables are generated.

Finally, if Y is in S_c, that is, if it is constrained in the QUIST query, then we generate a restriction term P(y) in the same way that we generated a restriction term P(x) for relation X in step 2.

Now suppose that relation Y had been introduced previously as a bridging relation. We do not need to link Y to X because every bridging relation is automatically linked to X. We do not need to introduce a new tuple variable y, because this has already been done. However, we do have to check whether Y is in S_{out}, the set of relations involved in the output. If so, we must transfer y from the set $Q_{\exists}$ where it was originally placed as a bridging variable, to the set Q_t, so it will end up in the target list. Also, we must generate a restriction term P(y) if Y is in S_c, the set of constrained relations.

This concludes the description of step 3 of the algorithm. When this has been done for all relations *in* S_{all}, *we are ready for the last step,* the actual generation of the relational query.

The target list of the query is simply the list of variables in Q_t. The qualification of the query is a conjunction of restriction terms P(y) for all relations Y in S_c, and join terms J(y,y') generated in the linking step. We can distinguish the P and J terms on whether they contain any variables in $Q_\exists$. Call the conjunctions of these terms $P(Q_\exists)$, $P(Q_t)$, $J(Q_\exists)$, and $J(Q_t)$. (Note that $J(Q_\exists)$ can have terms that refer to both a bound and a free variable.) We generate existential quantifiers for all variables in $Q_\exists$. They will be in the form

$$\exists v_1 \exists v_2 \ldots \exists v_k$$

if v_1 through v_k are the variables in $Q_\exists$. Let us abbreviate this as $\exists(Q_\exists)$. Then the relational query can be written as:

$$\{(Q_t) \mid P(Q_t) \wedge J(Q_t) \wedge \exists(Q_\exists)\,(P(Q_\exists) \wedge J(Q_\exists))\}$$

It really doesn't matter if we place the free variables within the quantifiers, so we can equivalently express the query as

$$\{(Q_t) \mid \exists(Q_\exists)\,(P(Q_{all}) \wedge J(Q_{all}))\}.$$

where of course the subscript "all" refers to all variables, free or bound. This emphasizes the similarity of the relational form to the original QUIST form: a simple conjunction of terms.

B.3 QUIST semantic rules and their relational counterparts

We now have a quite simple representation of a QUIST query in terms of the relational calculus. Next, we consider QUIST rules and transformations in terms of their relational counterparts.

For productions, we consider all the relations associated with the attributes involved in the rule. If we group the constraints by relation, we start with an expression like

$$\forall x,y_1,\ldots,y_n\; P(x) \wedge P_1(y_1) \wedge \ldots \wedge P_n(y_n) \rightarrow P(x)$$

but to this we must add the appropriate join terms to insure that the relations are properly linked. We select X as the root relation. The process of generating the join terms is then very much like the one previously described to build up a query. In particular, we may introduce additional variables that will appear existentially bound in the rule. Let R_r refer to variables that are present without being introduced for linkage purposes; $R_\exists$ refer just to those existentially bound variables; and R_{all} refer to all variables of either kind. Then, using the same kind of abbreviations as in our description of queries, the relational form of the production is

$$\forall R_r\,(\exists(R_\exists)\,J(R_{all}) \wedge P_r(R_r)) \rightarrow P(x).$$

A bounding rule obviously involves either one relation X or two relations X and Y. The case of one relation is quite simply $\forall x\; P(x)$ where P(x) is a comparison between two attributes of relation X. For two relations, the proper notion is $\forall x,y\; M(x,y) \rightarrow P(x,y)$. That is, given the proper mapping conditions between X and Y, possibly involving intervening relations (hence other existentially

bound variables), then some predicate P(x,y), the comparison between attributes of X and Y, holds. The form of the bounding rule turns out to be very similar to the form of the production. Recalling that M(x,y) may involve some existentially bound variables, we can write the bounding rule as

$$\forall x,y\,(\exists(R_{\exists})\,J(R_{all})) \rightarrow P(x,y).$$

B.4 QUIST transformations and their relational counterparts

QUIST transformations can now be seen to be a special case of the semantic equivalence transformations for relational queries described in Chapter 3. We illustrate this just for the case of a production. We start with some query in the form:

$$\{(Q_t) \mid \exists(Q_{\exists})\,(P(Q_{all}) \wedge J(Q_{all}))\}.$$

Suppose we wish to infer a constraint on the relation X using a production

$$\forall R_r\,(\exists(R_{\exists})\,J(R_{all}) \wedge P_r(R_r)) \rightarrow P(x).$$

The QUIST transformation corresponds to dropping all the universal quantifiers from this expression, leaving one in which R_r are free variables. In order to carry out the transformation, two conditions must be met. First, every relation ranged over by the variables in R_r must be a query relation ranged over by some variable in Q_{all}. This condition also guarantees that the query has the requisite join terms. Second, every restriction term in the conjunction $P_r(R_r)$ must be at least as strong as the corresponding restriction term in the conjunction $P(Q_{all})$. If these two conditions are met, then upon application of the logical schema

$$(A \wedge (A \rightarrow B)) \equiv (A \wedge B)$$

the constraint P(x) can be conjoined to the query expression, where x is the variable in the query that corresponds to x in the rule.

There is one additional case where the transformation can be made, that of *join introduction*. In that case, the relation X is not already part of the query even though all the antecedent conditions of the rule are met. We wish to add P(x) to the query. The only way to do so is to add in the necessary join terms to link X to the existing query relations. That is, we want to link X into the query in just the same way that we described above for constructing a query step by step. Obviously, x itself is not already in the target list, so x will be existentially bound in the query. Also, any intermediate relations needed to link in x will be existentially bound. Hence, we seek to introduce a conjunction of join terms that involve existentially bound variables, such as

$$\exists y_1,\ldots,\exists y_n,\exists x\; J(y,y_1) \wedge \ldots \wedge J(y_n,x)$$

where y is the variable that ranges over relation Y, the relation already in the query to which X can be linked. This expression can be conjoined to the original query without altering the answer if and only if every tuple in Y satisfies it; that is, if and only if the *structural* integrity constraint

$$\forall y\, \exists y_1,\ldots,\exists y_n,\exists x \mid J(y,y_1) \wedge \ldots \wedge J(y_n,x)$$

holds.

Bibliography

[ANSI75] ANSI/X3/SPARC Study Group on Data Base Management Systems.
Interim Report.
Bulletin of ACM SIGMOD 7(2), 1975.

[Astrahan75] Astrahan M. M. and Chamberlin D. D.
Implementation of a structured English query language.
Communications of the ACM 18(10):580-588, October, 1975.

[Astrahan80a] Astrahan M. M., Kim W., and Schkolnick M.
Evaluation of the System R access path selection mechanism.
Research Report RJ2797, IBM, April, 1980.

[Astrahan80b] Astrahan M. M. et al.
A history and evaluation of System R.
Research Report RJ2843, IBM, June, 1980.

[Barstow79] Barstow, D.R.
Knowledge-based Program Construction.
Elsevier North-Holland, New York, 1979.

[Blasgen77] Blasgen, M. and Eswaren, K.
Storage access in a relational database.
IBM Systems Journal 16(4):97-137, 1977.

[Brachman80] Brachman, Ronald J. and Smith, Brian C.
Special Issue on Knowledge Representation.
ACM SIGART Newsletter, February, 1980.

[Brodie78] Brodie, Michael L.
Specification and verification of data base semantic integrity.
Technical Report CSRG-91, Computer Systems Research Group, University of Toronto, April, 1978.

[Brodie80] Brodie, Michael L.
Data abstraction, databases, and conceptual modelling: an annotated bibliography.
Special Publication 500-59, National Bureau of Standards, May, 1980.

[Buchanan76] Buchanan, Bruce G.
Scientific theory formation by computer.
In Simon, J.C. (editor), *Proc. of Advanced Study Institute on Computer Oriented Learning Processes.* NATO, Noordhoff, Leyden, 1976.

[Carlson76] Carlson C. R. and Kaplan R. S.
A generalized access path model and its application to a relational data base system.
In *Proc. of ACM SIGMOD Conference*, pages 143-154. 1976.

[Chang78] Chang, C. L.
DEDUCE 2: Further investigations of deduction in relational databases.
In Gallaire, Herve and Minker, Jack (editors), *Logic and data bases*, pages 201-236. Plenum Press, 1978.

[Chen76] Chen, Peter.
The entity-relationship model -- toward a unified view of data.
ACM Transactions on Database Systems 1(1):9-36, March, 1976.

[Codd70] Codd, E. F.
A relational model for large shared data banks.
Communications of the ACM 13(6):377-387, June, 1970.

[Codd71] Codd, E. F.
Relational completeness of data base sublanguages.
In Rustin, Randall (editor), *Data Base Systems*, pages 65-98. Prentice-Hall, 1971.

[Comer79] Comer, D.
The ubiquitous B-tree.
ACM Computing Surveys 11(2):121-137, June, 1979.

[Couper72] Couper, A. D.
The geography of sea transport.
Hutchinson, London, 1972.

[Date77] Date, C. J.
An introduction to database systems (2nd ed.).
Addison Wesley, Reading, Massachusetts, 1977.

[Davis76] Davis, Randall.
Applications of meta level knowledge to the construction, maintenance and use of large knowledge bases.
PhD thesis, Stanford University, July, 1976.
Computer Science Department Report STAN-CS-76-552.

[Davis77] Davis, Randall.
Interactive transfer of expertise: acquisition of new inference rules.
In *Proc. Fifth Intl. Joint Conference on Artificial Intelligence*, pages 321-328. Cambridge, Massachusetts, August, 1977.

[Doyle78] Doyle, Jon.
Truth maintenance systems for problem solving.
Technical Report AI-TR-419, M.I.T. Artificial Intelligence Laboratory, January, 1978.

[ElMasri80a] El-Masri, Ramez and Wiederhold, Gio.
Properties of relationships and their representation.
In *Proc. National Computer Conference.* 1980.

[Elmasri80b] El-Masri, Ramez.
On the design, use, and integration of data models.
PhD thesis, Stanford University, June, 1980.

[Epstein78] Epstein, R., Stonebraker, M., and Wong, E.
Distributed query processing in a relational data base system.
Memo UCB/ERL M78/18, University of California at Berkeley, Electronics Research Lab, April, 1978.

[Eswaren75] Eswaren, Kapali and Chamberlin, Donald.
Functional specifications of a subsystem for database integrity.
In *Proc. First Intl. Conference on Very Large Data Bases*, pages 48-68. September, 1975.

[Feigenbaum71] Feigenbaum,E.A, Buchanan,B.G., and Lederberg,J.
On generality and problem solving: a case study using the DENDRAL program.
In Meltzer, B. and Michie, D. (editors), *Machine Intelligence 6*, pages 165-190. American Elsevier, New York, 1971.

[Fikes75] Fikes, Richard.
Deductive retrieval mechanisms for state description models.
In *Proc. Proc. Fourth Intl. Joint Conference on Artificial Intelligence*, pages 99-106. Tbilisi, USSR, September, 1975.

[Fry76] Fry,J.P. and Sibley,E.H.
Evolution of data-base management systems.
ACM Computing Surveys 8(1):7-42, March, 1976.

[Gallaire78] Gallaire, Herve and Minker, Jack (editors).
Logic and Data Bases.
Plenum Press, New York and London, 1978.

[Garvey76] Garvey,Thomas D.
Perceptual strategies for purposive vision.
Technical Note 117, SRI International Artificial Intelligence Center, September, 1976.

[Gotlieb75] Gotlieb, L.
Computing joins of relations.
In *Proc. of ACM SIGMOD Conference*, pages 55-63. 1975.

[Hall75] Hall, P.A.V.
Optimisation of a single relational expression in a relational data base system.
Technical Report UKSC 0076, IBM UK Scientific Centre, June, 1975.

[Hammer75] Hammer, Michael M. and McLeod, Dennis J.
Semantic integrity in a relational data base system.
In *Proc. First Intl. Conference on Very Large Data Bases*, pages 25-47. September, 1975.

[Hammer78] Hammer, Michael M. and Sarin, Sunil K.
Efficient monitoring of database assertions.
In *Supplement to Proc. of ACM SIGMOD Conference*, pages 38-49. 1978.

[Hayes-Roth78] Hayes-Roth, B. and Hayes-Roth, F.
Cognitive processes in planning.
Technical Report R-2366-ONR, RAND Corporation, December, 1978.

[Held75] Held,G.D., Stonebraker,M.R., and Wong,E.
INGRES - a relational data base system.
In *Proc. National Computer Conference.* 1975.

[Hendrix78] Hendrix, Gary G., et al.
Developing a natural language interface to complex data.
ACM Transactions on Database Systems 3(2):105-147, June, 1978.

[Janas79] Janas, Jurgen M.
How not to say NIL -- improving answers to failing queries in data base systems.
In *Proc. Sixth Intl. Joint Conference on Artificial Intelligence*, pages 429-434.
Tokyo, Japan, August, 1979

[Kant79] Kant, Elaine.
Efficiency considerations in program synthesis: a knowledge based approach.
PhD thesis, Stanford University Computer Science Department, September, 1979.
Technical Report STAN-CS-79-755.

[Kaplan79] Kaplan, S. Jerrold.
Cooperative responses from a portable natural language data base query system.
PhD thesis, Dept. of Computer and Information Science, University of Pennsylvania, 1979.

[Katz80] Katz,R.H. and Wong,E.
An access path model for physical database design.
In *Proc. of ACM SIGMOD Conference*, pages 22-28. May, 1980.

[Kellog78] Kellog, Charles, Klahr, Phillip, and Travis, Larry.
Deductive planning and pathfinding for relational data bases.
In Gallaire, Herve and Minker, Jack (editors), *Logic and data bases*, pages 179-200.
Plenum Press, 1978.

[Kent78] Kent, William.
Data and reality.
North-Holland, Amsterdam, 1978.

[Kim79] Kim, Won.
Relational database systems.
ACM Computing Surveys 3(11):185-212, September, 1979.

[King79] King, Jonathan J.
Exploring the use of domain knowledge for query processing efficiency.
Heuristic Programming Project Technical Report HPP-79-30, Stanford University Computer Science Department, December, 1979.

[Klahr78] Klahr, Philip.
Planning techniques for rule selection in deductive question-answering.
In Waterman, D.A. and Hayes-Roth, F. (editors), *Pattern-Directed Inference Systems*, pages 223-239. Academic Press, 1978.

[Konolige81] Konolige, Kurt.
The database as a model: a metatheoretic view.
Technical Report, SRI International Artificial Intelligence Center, forthcoming 1981.

[Lenat76] Lenat, Douglas B.
AM: an artificial intelligence approach to discovery in mathematics as heuristic search.
PhD thesis, Stanford University, July, 1976.
Stanford Artificial Intelligence Laboratory Memo 286.

[Lenat79] Lenat, D. B., Hayes-Roth, F., Klahr, P.
Cognitive economy.
Technical Report N-1185-NSF, Rand Corporation, June, 1979.

[Lloyds78] Lloyds Register of Ships 1978-1979.

[McLeod76] McLeod, Dennis J.
High level expression of semantic integrity specifications in a relational data base system.
Technical Report 165, MIT Laboratory for Computer Science, September, 1976.

[McLeod78] McLeod, Dennis J.
A semantic data base model and its associated structured user interface.
PhD thesis, MIT, August, 1978.

[McLeod81] McLeod, Dennis J. and Smith, John Miles.
Abstraction in databases.
SIGPLAN Notices 16(1), January, 1981.

[McSkimin77] McSkimin, James R. and Minker, Jack.
The use of a semantic network in a deductive question answering system.
In *Proc. Fifth Intl. Joint Conference on Artificial Intelligence*, pages 50-58. 1977.

[Moore79] Moore, Robert C.
Handling complex queries in a distributed database.
Technical Note 170, SRI International Artificial Intelligence Center, October, 1979.

[Nicolas78a] Nicolas, J. M. and Gallaire, H.
Data base: theory vs.interpretation.
In Gallaire, Herve and Minker, Jack (editors), *Logic and Data Bases*, pages 33-54.
Plenum Press, New York and London, 1978.

[Nicolas78b] Nicolas, J. M. and Yazdanian, K.
Integrity checking in deductive databases.
In Gallaire, Herve and Minker, Jack (editors), *Logic and Data Bases*, pages 325-344.
Plenum Press, New York and London, 1978.

[Nilsson71] Nilsson, Nils. J.
Problem-Solving Methods in Artificial Intelligence.
McGraw Hill, New York, 1971.

[Pecherer75] Pecherer, R. M.
Efficient evaluation of expressions in a relational algebra.
In *Proc. ACM Pacific 75 Conference*, pages 44-49. April, 1975.

[Pirotte78] Pirotte, Alain.
High level data base query languages.
In Gallaire, Herve and Minker, Jack (editors), *Logic and Data Bases*, pages 409-436.
Plenum Press, New York and London, 1978.

[Reiter78] Reiter, Raymond.
Deductive question-answering on relational data bases.
In Gallaire, Herve and Minker, Jack (editors), *Logic and data bases*, pages 149-177.
Plenum Press, New York and London, 1978.

[Rothnie75] Rothnie, J. B.
Evaluating inter-entry retrieval expressions in a relational database management system.
In *Proc. 1975 National Computer Conference*, pages 417-423. 1975.

[Roussopoulos77] Roussopoulos, Nicholas D.
A semantic network model of data bases.
PhD thesis, Univ. of Toronto Computer Science Department, April, 1977.

[Sacerdoti77] Sacerdoti, E.D.
A structure for plans and behavior.
American Elsevier, New York, 1977.

[Sagalowicz77] Sagalowicz, Daniel.
IDA: an intelligent data access program.
In *Third Intl. Conference on Very Large Data Bases.* October, 1977.

[Schmid75] Schmid, H. A. and Swenson, J.R.
On the semantics of the relational data model.
In *Proc. of ACM SIGMOD Conference*. May, 1975.

[Selinger79] Selinger, P. Griffiths et. al.
Access path selection in a relational database management system.
In *Proc. of ACM SIGMOD Conference*, pages 23-34. May, 1979.

[Shaw80] Shaw, D.
Knowledge-based retrieval on a relational database machine.
PhD thesis, Stanford University, August, 1980.

[Shipman79] Shipman, David W.
The functional data model and the data language DAPLEX.
In *Proc. of ACM SIGMOD Conference (Supplement)*, pages 1-19. May, 1979.

[Shortliffe76] Shortliffe, E. H.
MYCIN: Computer-Based Medical Consultations.
American Elsevier, 1976.

[Smith75] Smith, J. M. and Chang, P.
Optimizing the performance of a relational algebra data base interface.
Communications of the ACM 10(18):568-579, October, 1975.

[Smith78] Smith, Brian.
Levels, layers, and planes: the framework of a theory of knowledge representation semantics.
Master's thesis, MIT Artificial Intelligence Laboratory, 1978.

[Sproull77] Sproull, Robert F.
Strategy construction using a synthesis of heuristic and decision-theoretic methods.
PhD thesis, Stanford University, July, 1977.

[Stefik80] Stefik, M.J.
Planning with constraints.
PhD thesis, Stanford University, January, 1980.
STAN-CS-80-784.

[Stonebraker75] Stonebraker, Michael.
Implementation of integrity constraints and views by query modification.
In *Proc. of ACM SIGMOD Conference*, pages 65-78. May, 1975.

[Stonebraker76] Stonebraker, Michael et. al.
The design and implementation of INGRES.
ACM Transactions on Database Systems 3(1):189-222, September, 1976.

[Stonebraker80] Stonebraker, Michael.
Retrospection on a database system.
ACM Transactions on Database Systems 5(2):225-240, June, 1980.

[Taylor76] Taylor, R. W. and Frank, R. L.
CODASYL data-base management systems.
ACM Computing Surveys 1(8):67-104, March, 1976.

[Teitelman78] Teitelman, Warren.
Interlisp Reference Manual
Xerox Palo Alto Research Center, Palo Alto, California, 1978.

[Tsichritizis76] Tsichritizis, D. C. and Lochovsky, F. H.
Hierarchical data-base management.
ACM Computing Surveys 1(8):105-124, March, 1976.

[Ullman80] Ullman, Jeffery D.
Principles of Database Systems.
Computer Science Press, 1980.

[Vanmelle79] van Melle, William.
A domain-independent production-rule system for consultation programs.
In *Proc. Sixth Intl. Joint Conference on Artificial Intelligence*, pages 923-925.
Tokyo, Japan, August, 1979.

[Weyhrauch80] Weyhrauch, Richard.
Prolegomena to a theory of mechanized formal reasoning.
Artificial Intelligence 13(1-2):133-170, 1980.

[Wiederhold77] Wiederhold, Gio.
Database design.
McGraw-Hill, New York, 1977.

[Wilson80] Wilson, Gerald A.
A conceptual model for semantic integrity checking.
In *Proc. Sixth Intl. Conference on Very Large Data Bases.* September, 1980.

[Wong76] Wong, Harry K.T. and Mylopoulos, John.
Two views of data semantics: a survey of data models in artificial intelligence and database management.
A.I. Memo, University of Toronto Department of Computer Science, December, 1976.

[Wong76b] Wong, E. and Youssefi, K.
Decomposition -- a strategy for query processing.
ACM Transactions on Database Systems 3(1):223-241, September, 1976.

[Yao78] Yao, S. Bing and De Jong, David.
Evaluation of access paths in a relational database system.
Technical Report 280, Purdue University Department of Computer Sciences, August, 1978.

[Yao79] Yao, S. Bing.
Optimization of query evaluation algorithms.
ACM Transactions on Database Systems 2(4):133-155, June, 1979.

[Youssefi78] Youssefi, Karel A. Allen.
Query processing for a relational database system.
PhD thesis, University of California, Berkeley, January, 1978.
Electronics Research Laboratory Memorandum UCB/ERL M78/3.

[Youssefi79] Youssefi, Karel A. Allen and Wong, Eugene.
Query processing in a relational database management system.
In *Fifth Intl. Conference on Very Large Data Bases*, pages 409-417. 1979.